Certificate

This is to certify that

Mr./~~Ms~~ **NIRAJKUMAR ITALIYA** Of class **CE_Sem-5** Division **A2** Enrollment No **190130107041** Has satisfactorily completed his/~~her~~ term work in **ADA(*3150703)*** Subjec t for the term ending in November **2021-2022**.

Date:- **23/09/2021**

Government Engineering College, Gandhinagar

Computer Engineering

B.E. Semester V

(AY 2021-22)

SUBJECT

ANALYSIS AND DESIGN OF ALGORITHMS (3150703)

Student Details:

Enrolment No. :	190130107041
Full Name : (Surname Mid. First Name)	Italiya NirajKumar Vijaybhai
Sub-Division:	A2
Email Id:	nirajitaliya2019@gecg28.ac.in
Contact Number:	8140810073

Student Details:

Student:	
Subject Coordinator/Lab Faculty:	
HOD Signature:	

Vision and Mission

Institute (GECG):

Vision:	To be a premier engineering institution, imparting quality education for innovative solutions relevant to society and environment.
Mission:	● To develop human potential to its fullest extent so that intellectual and innovative engineers can emerge in a wide range of professions. ● To advance knowledge and educate students in engineering and other areas of scholarship that will best serve the nation and the world in future. ● To produce quality engineers, entrepreneurs and leaders to meet the present and future needs of society as well as the environment.

Department (CE):

Vision:	To achieve excellence for providing value based education in Computer Engineering through innovation, teamwork and ethical practices.
Mission:	● To produce computer science and engineering graduates according to the needs of industry, government, society and scientific community. ● To develop partnership with industries, government agencies and R and D Organizations ● To motivate students/graduates to be entrepreneurs. ● To motivate students to participate in reputed conferences, workshops, symposiums, seminars and related technical activities

Subject Specific Links:

Syllabus	3150703_ADA_NEW_Syllabus.pdf
Lecture Notes:	DPK-ADA-Lecture-Notes
Question Bank:	DS Question Bank.docx
Submission: Assignments Practical	GECG_ADA_Assignments and Practicals.docx

Exams Information:

Exams	Marks	Tentative Date
Mid Sem	30	Aug-2021
Internal Viva	20	Sept-2021
GTU Exam	70	Nov-2021
External Viva	30	Oct-2021

Submission:
link:
https://docs.google.com/forms/d/1LSvoNEcdl7wpAa0VAD3o3qox4Zv4PhaB5_HyJLBGwz8/edit

Submission	By Date	Practical	Assignments (Hand Written Only)
1			
2			
3			
Complete Submission	One week before Internal Viva	All	All

Index-1

Sr.No.	Assignment	Date	Page No.
1			
2			
3			
4			
5			
6			
7			
8			
9			

Index-2

Sr.No.	Practical	Date	Page No.
1	Implementation and Time analysis of sorting algorithms. Bubble sort, Selection sort, Insertion sort, Merge sort and Quicksort.	21/6/2021	5
2	Implementation and Time analysis of linear and binary search algorithm.	28/6/2021	28
3	Implementation the min and max heap sort algorithm.	28/6/2021	36
4	Implementation and Time analysis factorial program using iterative and recursive method.	5/7/2021	49
5	Implement Knapsak Problem using Greedy Method.	12/7/2021	54
6	Implementation the chain matrix multiplication using dynamic programming.	19/7/2021	57
7	Implementation of making a change problem using dynamic programming	26/7/2021	59
8	Implementation of a knapsack problem using greedy algorithm.	2/8/2021	60
9	Implementation of Graph and Searching (DFS and BFS).	9/8/2021	63

Index-2

10	Implementation prism's Algorithm for minimum spanning tree.	16/8/2021	66
11	Implementation Kruskal's Algorithm for minimum spanning tree.	6/9/2021	68
12	Implementat lCS Problem	13/9/2021	71

Practical 1

Implementation and Time analysis of sorting algorithms. Bubble sort, Selection sort, Insertion sort, Merge sort and Quicksort.

Bubble Sort

Algorithm:

```
begin BubbleSort(list,n)
        for pass=0, pass<n-1, pass++
            for i=0, i<=n-1-pass, i++
                if list[i] > list[i+1]
                swap(list[i], list[i+1])
                end if
            end for
        end for
            return list
end BubbleSort
```

Time Complexity:

Here there are two for loops, So time complexity is $O(n^2)$.

Space Complexity:

Here there is one array of size n, So space complexity is $O(n)$.

Recurrence Relation:

T(n)=T(n-1)+n

Similarly,

T(n-1)=T(n-2)+n-1

So on till,

= T(1) + (2+3+…+n)

=T(0) + (1+2+3+…+n)

= T(0) + n*(n+1)/2

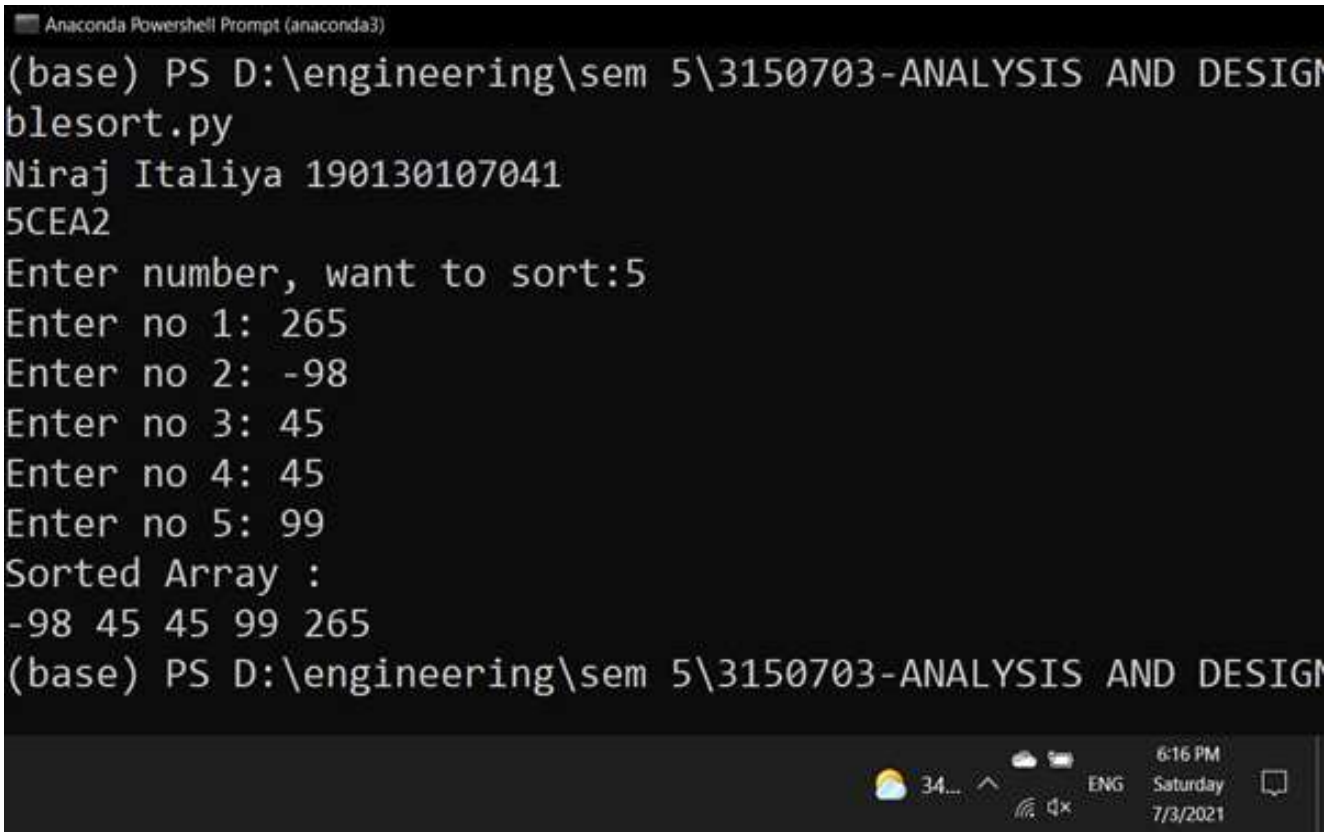
= O(n^2)

Implementation :

```
print(“Niraj Italiya 190130107041”)
print("5CEA2")

def bubbleSort(arr):
    n = len(arr)
    for i in range(n-1):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]

size = int(input("Enter number, want to sort:"))
```

```
arr = []
for i in range(size):
        value = int(input("Enter no {i}: ".format(i=i+1)))
        arr.append(value)
bubbleSort(arr)

print("Sorted Array : ")
for i in range(size):
        print(arr[i], end = " ")
```

Output:



Conclusion:

- To sort a list of ‘n’ by bubble sort it will take n-1 iteration.
- After each pass the heaviest element is on the rightmost pos^{n}.
- Worst case would be when array is in descending order.
- Algorithm can be improved by adding a flag to stop the inner loop if no swaps occurred.

Selection Sort

Algorithm:

```
Begin selection sort (list,n)
        for i = 1 to n - 1
            min = i
          for j = i+1 to n
                if list[j] < list[min] then
                    min = j;
                end if
            end for
            if indexMin != i then
                swap list[min] and list[i]
            end if
        end for
end selection sort
```

Time Complexity:

Here there are two for loops, So time complexity is $O(n^2)$.

Space Complexity:

Here there is one array of size n, So space complexity is $O(n)$.

Recurrence Relation:

T(n)=T(n-1)+n

Similarly,

T(n-1)=T(n-2)+n-1

So on till,

= T(1) + (2+3+…+n)

=T(0) + (1+2+3+…+n)

= T(0) + n*(n+1)/2

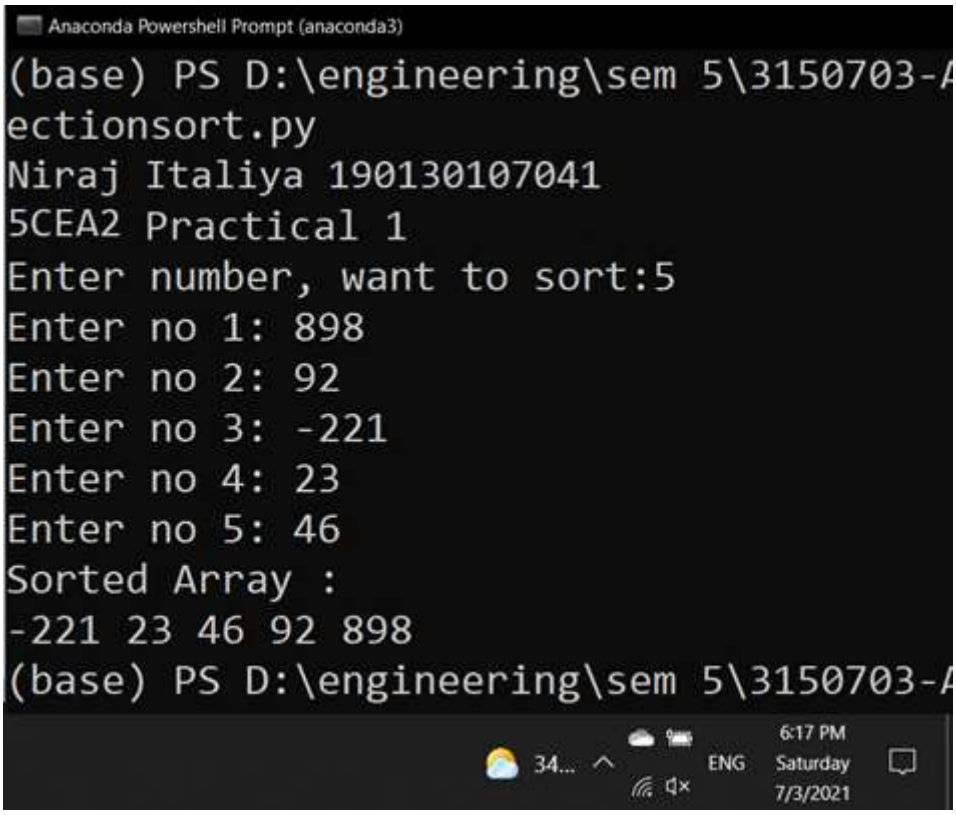
= $O(n^2)$

Implementation :

```
print(“Niraj Italiya 190130107041”)
print("5CEA2 Practical 1")

def selectionSort(arr):
        n = len(arr)
        for i in range(n):
                min_in = i
                for j in range(i+1, n):
                if arr[min_in] > arr[j]:
                                min_in = j
                arr[i], arr[min_in] = arr[min_in], arr[i]
```

```
size = int(input("Enter number, want to sort:"))


arr = []
for i in range(size):
        value = int(input("Enter no {i}: ".format(i=i+1)))
        arr.append(value)
selectionSort(arr)


print("Sorted Array : ")
for i in range(size):
        print(arr[i], end = " ")
```

Output:



Conclusion:

- Selection sort is a combination of searching and sorting.
- After each pass the smallest element is on the leftmost pos^{n}.
- Worst case would be when array is in descending order
- To sort a list of ‘n’ by selection sort it will take n-1 passes.
- Best case worse case avg case are all O(n^2).

Insertion Sort

Algorithm:

```
Begin insertion sort (list,n)
        for j = 2 to n
            key = list[j]
            i = j-1
            while i > 0 and list[i] > key
                list[i+1] = list[i]
                i = i-1
            end while
        end for
end insertion sort
```

Time Complexity:

Here there are two for loops, So time complexity is $O(n^2)$.

Space Complexity:

Here there is one array of size n, So space complexity is $O(n)$.

Recurrence Relation:

T(n)=T(n-1)+n

Similarly,

T(n-1)=T(n-2)+n-1

So on till,

= T(1) + (2+3+…+n)

=T(0) + (1+2+3+…+n)

= T(0) + n*(n+1)/2

= O(n^2)

Implementation :

```
print(“Niraj Italiya 190130107041”)
print("5CEA2 Practical 1")

def insertionSort(arr):
    n = len(arr)
    for i in range(n):
        k = arr[i]
        j = i-1

        while j >=0 and k < arr[j]:
            arr[j+1] = arr[j]
```

```
                j = j-1
            arr[j+1] = k


size = int(input("Enter number, want to sort:"))


arr = []
for i in range(size):
    value = int(input("Enter no {i}: ".format(i=i+1)))
    arr.append(value)
insertionSort(arr)
print("Sorted Array : ")
for i in range(size):
    print(arr[i], end = " ")
```

Output:

```
(base) PS D:\engineering\sem 5\3150703-ANALYSIS AND
3_Insertionsort.py
Niraj Italiya 190130107041
5CEA2 Practical 1
Enter number, want to sort:5
Enter no 1: 985
Enter no 2: 56
Enter no 3: -45
Enter no 4: -87
Enter no 5: 23
Sorted Array :
-87 -45 23 56 985
(base) PS D:\engineering\sem 5\3150703-ANALYSIS AND
```

Conclusion:

- To sort a list of ‘n’ by insertion sort we divide it into two virtual arrays, a sorted array and an unsorted array.
- After each pass the selected element gets placed into the correct place in the sorted array.
- Worst case would be when array is in descending order.

Quick Sort

Algorithm:

```
Begin partitionFunc(left, right, pivot)
  leftPointer = left
  rightPointer = right – 1
  while True do
    while A[++leftPointer] < pivot do
      //do-nothing
    end while
            while rightPointer > 0 && A[--rightPointer]
> pivot
      do
    end while
    if leftPointer >= rightPointer
      break
    else
      swap leftPointer,rightPointer
    end if
  end while
swap leftPointer,right
  return leftPointer
      end partitionFunc
```

Time Complexity:

Here there are two for loops, So time complexity is

O(n (log n)).

Space Complexity:

Here there is one array of size n, So space complexity is O(n (log n)).

Recurrence Relation:

T(n)=T(n-1)+n

[T(n)=O(n)+T(0)+T(n-1)=O(n)+T(n-1)]

Similarly,

T(n-1)=T(n-2)+n-1

So on till,

= T(1) + (2+3+…+n)

=T(0) + (1+2+3+…+n)

= T(0) + n*(n+1)/2

= $O(n^2)$

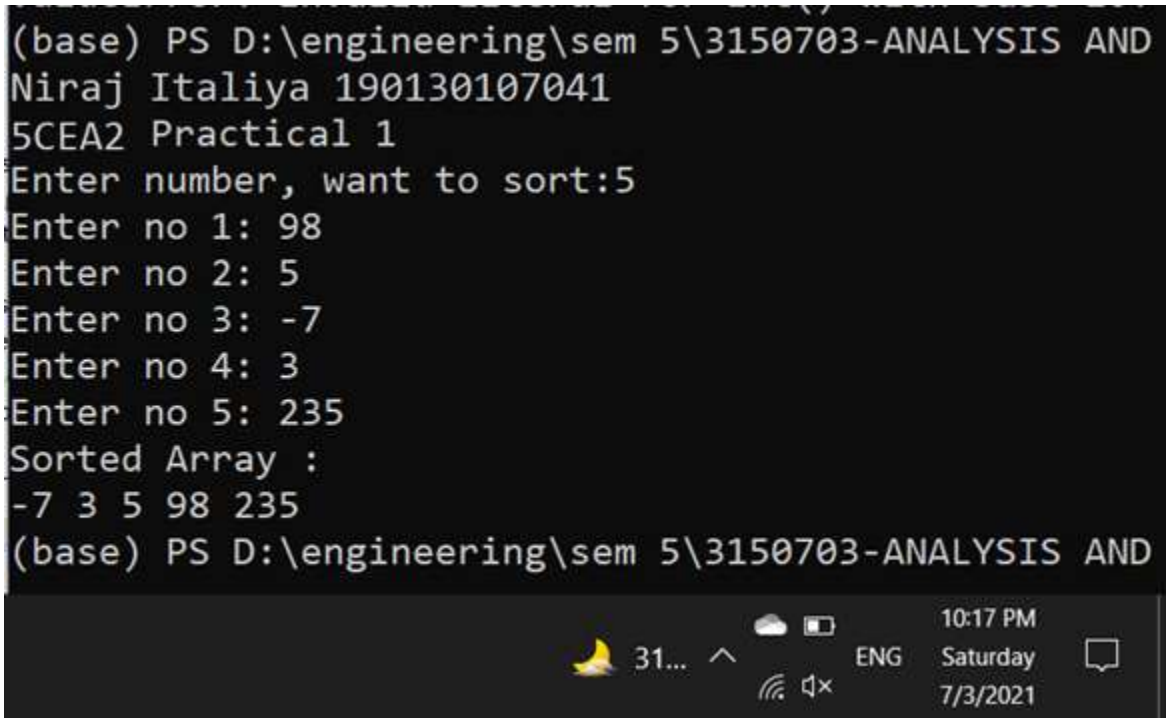
Implementation :

```
print(“Niraj Italiya 190130107041”)
print("5CEA2 Practical 1")


def partition(arr, low, high):
    i = (low-1)
    pivot = arr[high]
```

```
 for j in range(low, high):
        if arr[j] <= pivot:
            i = i+1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i+1], arr[high] = arr[high], arr[i+1]
    return (i+1)

def quickSort(arr, low, high):
    if len(arr) == 1:
        return arr
    if low < high:
        pi = partition(arr, low, high)
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)




size = int(input("Enter number, want to sort:"))

arr = []
for i in range(size):
        value = int(input("Enter no {i}: ".format(i=i+1)))
        arr.append(value)0
```

```
n = len(arr)
quickSort(arr, 0, n-1)
print("Sorted Array : ")
for i in range(size):
        print(arr[i], end = " ")
```

Output:



Conclusion:

- To sort a list of ‘n’ by quicksort we select a pivot and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot
- Sub arrays are sorted recursively
- Worst case would be when pivot is at the beginning pr the end , i.e the array is sorted
- Best case occurs when median element is selected as pivot

Merge Sort

Algorithm:

Begins mergeSort(arr[], l, r)

If r > l

Find the middle point to divide the array into two halves:

middle m = l+ (r-l)/2

Call mergeSort for first half:

Call mergeSort(arr, l, m)

Call mergeSort for second half:

Call mergeSort(arr, m+1, r)

Merge the two halves sorted in step 2 and 3:

Call merge(arr, l, m, r)

Time Complexity:

Here there are two for loops, So time complexity is O(n (log n)).

Space Complexity:

Here there is one array of size n, So space complexity is O(n).

Space Complexity:

Here there is one array of size n, So space complexity is O(n (log n)).

Recurrence Relation:

$T(n) = 2T(n/2) + \theta(n)$

Similarly,

$T(n)=2T(n/2)+n$,

$=4T(n/4)+2n$,

$=2^kT(n/2^k)+kn$

(Where , no. of reoccurrences till $T(n/2)=T(1)$ is $\log_2(n)$ so, substituting $k= \log_2(n)$, we get)

=**O(n log n)**

Implementation :

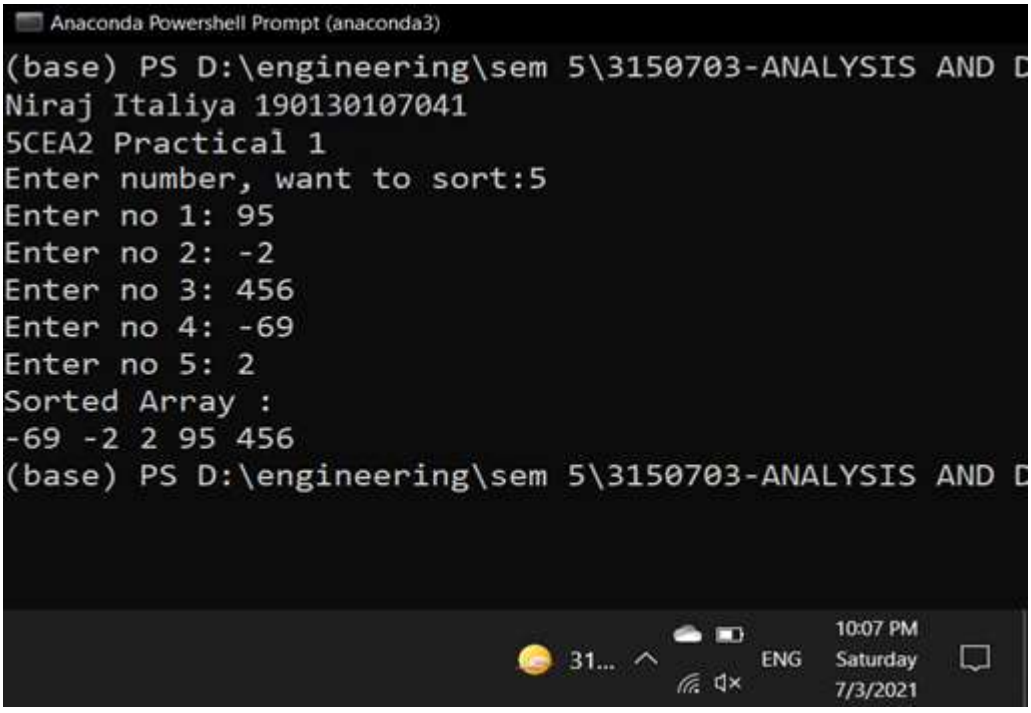
```
print("Niraj Italiya 190130107041")
print("5CEA2 Practical 1")

def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r- m
```

```
L = [0] * (n1)
R = [0] * (n2)

for i in range(0 , n1):
    L[i] = arr[l + i]

for j in range(0 , n2):
    R[j] = arr[m + 1 + j]

i = 0
j = 0
k = l
while i < n1 and j < n2 :
    if L[i] <= R[j]:
        arr[k] = L[i]
        i += 1
    else:
        arr[k] = R[j]
        j += 1
    k += 1
```

```
 while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

def mergeSort(arr,l,r):
    if l < r:

        m = (l+(r-1))//2
        mergeSort(arr, l, m)
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)

size = int(input("Enter number, want to sort:"))

arr = []
for i in range(size):
        value = int(input("Enter no {i}: ".format(i=i+1)))
        arr.append(value)
```

```
n=len(arr)
mergeSort(arr, 0, n-1)

print("Sorted Array : ")
for i in range(size):
    print(arr[i], end = " ")
```

Output:

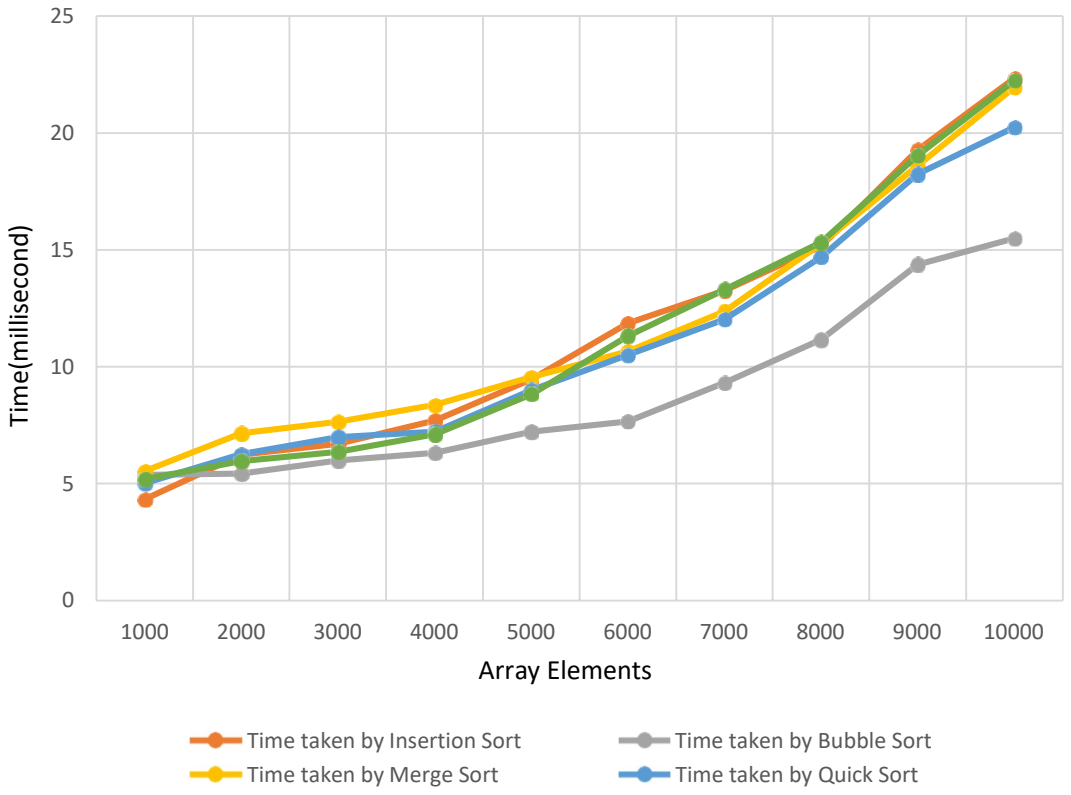


Conclusion:

- To sort a list of ‘n’ by insertion sort we divide it into two virtual arrays, a sorted array and an unsorted array.
- After each pass the selected element gets placed into the correct place in the sorted array.
- Worst case would be when array is when the two sub arrays are skipped numbers eg. {1,5} and {3,7}
- Algorithm has same complexity for best avg and worst case.

Array Elements	Time taken by Insertion Sort	Time taken by Bubble Sort	Time taken by Merge Sort	Time taken by Quick Sort	Time taken by Selection Sort
1000	4.322	5.369	5.531	5.018	5.184
2000	6.235	5.421	7.147	6.254	5.96
3000	6.695	5.989	7.635	6.985	6.359
4000	7.698	6.315	8.365	7.219	7.102
5000	9.465	7.215	9.555	8.994	8.823
6000	11.852	7.658	10.658	10.502	11.312
7000	13.259	9.315	12.364	12.021	13.299
8000	15.155	11.159	15.265	14.697	15.319
9000	19.2665	14.368	18.598	18.23	19.032
10000	22.326	15.489	21.951	20.232	22.236

Graph



What have you learned from this experiment?

(Tick ✓ the appropriate)	Learning of	Write an appropriate answer in one or two words.
✔	Theoretical Concept	Very Well
✔	Problem Solving	Improvement is good.
✔	Coding Style	Learn new method to write program.
	Testing	
✔	Way to answer a question in exam	Perfect

Practical 2

Implementation and Time analysis of linear and binary search algorithm.

Linear Search

Algorithm:

```
begin linear_search (list, value)
        for each item in the list
             if match item == value
                  return the item's location
            end if
       end linear_search
```

Time Complexity:

Here we are searching the entire array for every element, So time complexity is O(n).

Space Complexity:

Here we are returning only the index of one element, So space complexity is O(1).

Derivation :

For linear search simply iterate through the array and if the element is not present then that is the worst case ,

```
for(i=0;i<n;i++)
{
    If(a[i]==key)
    {
        printf(“key found at %d”,i);
    }
}
```

This gets exceuted 1+(n+1)+n+(n-1) times so overall we get,

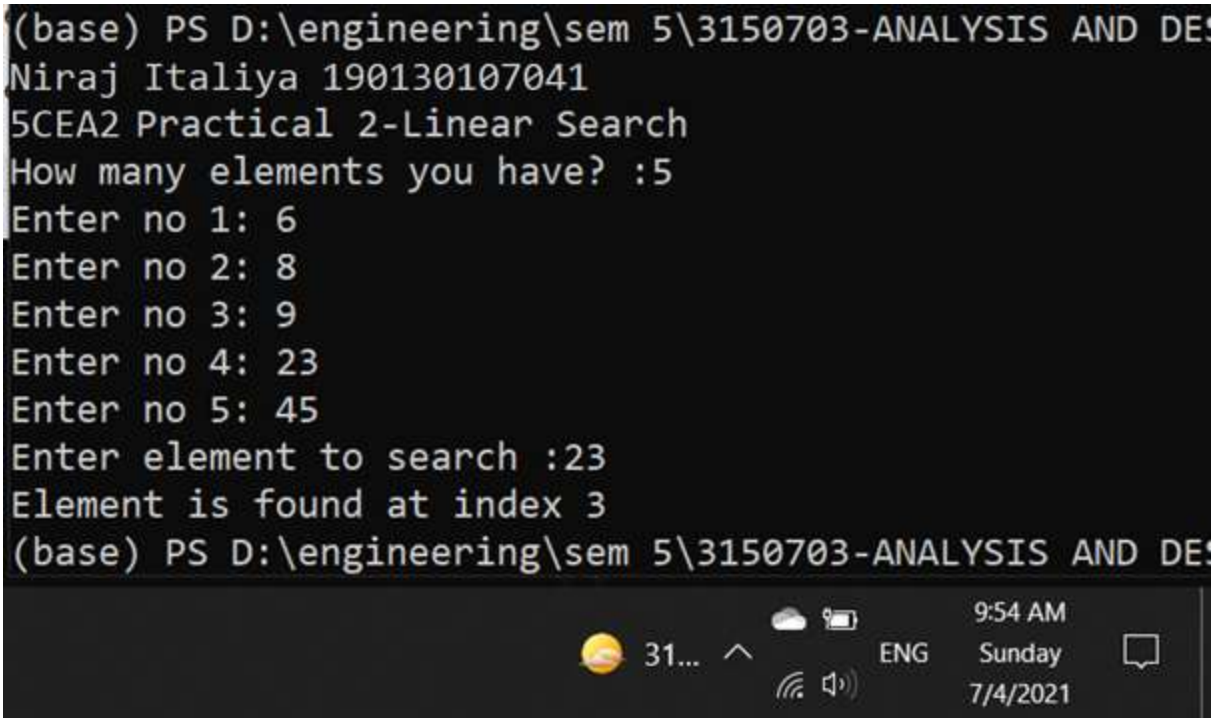
=**O(n)**

Implementation :

```
print(“Niraj Italiya 190130107041”)
 print("5CEA2 Practical 2-Linear Search")

def LinearSearch(arr, x):
    for i in range(len(arr)):

        if arr[i] == x:
            return i
    return -1
```

```
size = int(input("How many elements you have? :"))

arr = []
for i in range(size):
        value = int(input("Enter no {i}: ".format(i=i+1)))
        arr.append(value)

x = int(input("Enter element to search :"))
result = LinearSearch(arr, x)

if result != -1:
        print("Element is found at {i}".format(i=result))
else:
        print("Element is not found.")
```

Output:



Conclusion:

- To search a list of ‘n’ by linear search start from index 0 and go till n and see if you found the key element.
- Worst case would be when key element will not be present in the array.

Binary Search

Algorithm:

```
begin binary_search
  Set low = 1
  Set high = n
  while key not found
    if high < low
      EXIT: key does not present in array.
    set mid = low + ( high – low ) / 2
    if A[mid] < key
      set low = mid + 1
    if A[mid] > key
      set high = mid - 1
    if A[mid] = key
      EXIT: key found at location mid
  end while

end binary_search
```

Time Complexity:

Here we are searching the entire array for every element, So time complexity is O(log n).

Space Complexity:

Here we are returning only the index of one element, So space complexity is O(1).

Reoccurrence Relation:

$T(n) = T(n/2) + 1$

$a=1\ b=2\ k=0\ p=0$

$a=b^k$

$T(n)= \theta(n^{\log_b a} \log^{p+1} n)$

$= \theta(n^{\log_2 1} \log^{0+1} n)$

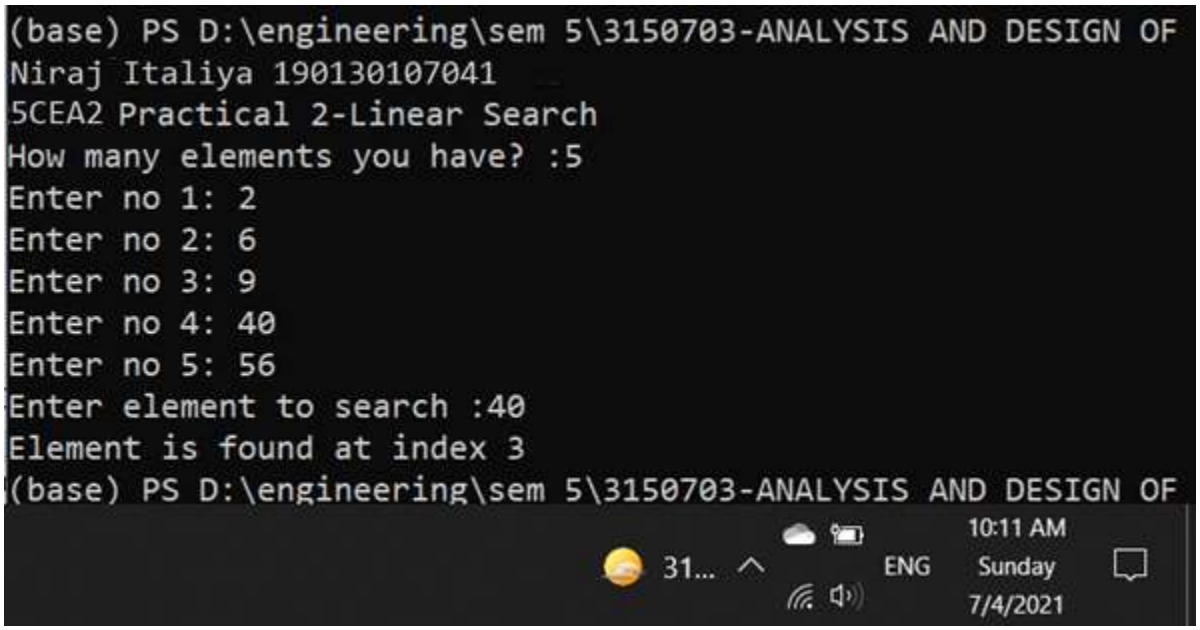
$= \boldsymbol{\theta (\log n)}$

Implementation :

```
print("Niraj Italiya 190130107041")
print("5CEA2 Practical 2-Linear Search")


def binarySearch(arr, low, high, x):

    if high >= low:

        mid = (high + low) // 2

        if arr[mid] == x:
            return mid
```

```
elif arr[mid] > x:
        return binarySearch(arr, low, mid - 1, x)
    else:
        return binarySearch(arr, mid + 1, high, x)
  else:
    return -1

size = int(input("How many elements you have? :"))

arr = []
for i in range(size):
    value = int(input("Enter no {i}: ".format(i=i+1)))
    arr.append(value)
n=len(arr)

x = int(input("Enter element to search :"))
result = binarySearch(arr, 0, n-1, x)

if result != -1:
    print("Element is found at index {i}".format(i=result))
else:
    print("Element is not found.")
```

Output:



Conclusion:

- To search a list of 'n' by binary search we divide it into two parts, and based on rel^n b/w key and mid we go to left or right sub-halves.
- After each pass the size of array reduces by half.
- Worst case would be when array is when the key element is (not in the array/in the last position relative to the search)
- Algorithm has constant complexity for best case O(1).

What have you learned from this experiment?

(Tick ✓ the appropriate)	Learning of	Write an appropriate answer in one or two words.
✓	Theoretical Concept	Got all the concepts.
✓	Problem Solving	Learnt searching in various data.
	Coding Style	
	Testing	
	Way to answer a question in exam	

Practical 3

Implementation the min and max heap sort algorithm.

Max Heap

Algorithm:

begin maxHeap(list,n)

Arr[(i-1)/2] Returns the parent node.
Arr[(2*i)+1] Returns the left child node.
Arr[(2*i)+2] Returns the right child node.

end maxHeap

- **getMax():** It returns the root element of Max Heap. Time Complexity of this operation is O(1).
- **extractMax():** Removes the maximum element from MaxHeap. Time Complexity of this Operation is O(Log n) as this operation needs to maintain the heap property (by calling heapify()) after removing root.
- **insert():** Inserting a new key takes O(Log n) time. We add a new key at the end of the tree. If new key is smaller than its parent, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.

Time Complexity:

Here there are two for loops, So time complexity is O(nlog n).

Space Complexity:

Here there is one array of size n, So space complexity is O(1).

Recurrence Relation:

T(n)= T(n-1) + O(log n)

Similarly,

T(n) = O(n) + n O(log n)

= **O(n log n)**

Implementation :

```
import sys

print("190130107041 Niraj Italiya")
print("5CEA2 Practical 3-Max Heap")

class MaxHeap:

   def__init__(self, maxsize):

      self.maxsize = maxsize
      self.size = 0
      self.Heap = [0] * (self.maxsize + 1)
      self.Heap[0] = sys.maxsize
      self.FRONT = 1
```

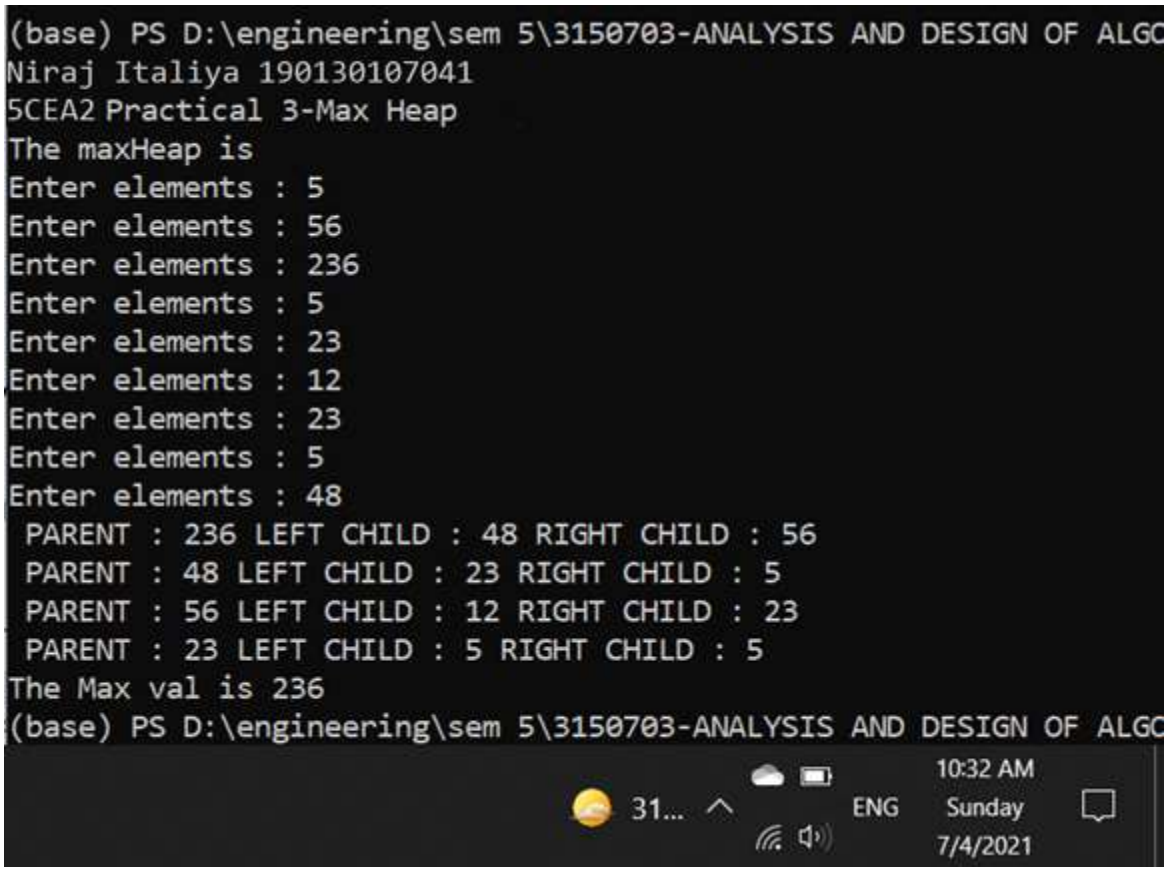
```
def parent(self, pos):
        return pos //2

def leftChild(self, pos):
         return 2 * pos

 def rightChild(self, pos):
        return (2 * pos) + 1

    def isLeaf(self, pos):
        if pos >= (self.size//2) and pos <= self.size:
           return True
        return False

    def swap(self, fpos, spos):

        self.Heap[fpos], self.Heap[spos] = (self.Heap[spos],
                                            self.Heap[fpos])

    def maxHeapify(self, pos):

        if not self.isLeaf(pos):
           if (self.Heap[pos] < self.Heap[self.leftChild(pos)] or
```

```
self.Heap[pos] < self.Heap[self.rightChild(pos)]):

            if (self.Heap[self.leftChild(pos)] >
               self.Heap[self.rightChild(pos)]):
               self.swap(pos, self.leftChild(pos))
               self.maxHeapify(self.leftChild(pos))

            else:
               self.swap(pos, self.rightChild(pos))
               self.maxHeapify(self.rightChild(pos))

  def insert(self, element):

     if self.size >= self.maxsize:
        return
     self.size += 1
     self.Heap[self.size] = element

     current = self.size

     while (self.Heap[current] >
          self.Heap[self.parent(current)]):
        self.swap(current, self.parent(current))
```

```
 current = self.parent(current)

    def Print(self):

        for i in range(1, (self.size // 2) + 1):
            print(" PARENT : " + str(self.Heap[i]) +
                  " LEFT CHILD : " + str(self.Heap[2 * i]) +
                  " RIGHT CHILD : " + str(self.Heap[2 * i + 1]))

    def extractMax(self):

        popped = self.Heap[self.FRONT]
        self.Heap[self.FRONT] = self.Heap[self.size]
        self.size -= 1
        self.maxHeapify(self.FRONT)

        return popped

if__name__== "__main__":

    print('The maxHeap is ')

    maxHeap = MaxHeap(15)
```

```
for i in range(0, 9):
    x = int(input("Enter elements : "))
    maxHeap.insert(x)

maxHeap.Print()

print("The Max val is " + str(maxHeap.extractMax()))
```

Output:



Conclusion:

- To sort a list of max heap sort it will take n iteration.
- After each pass the heaviest element is on the rightmost pos^{n}.
- Worst case would be when array is in descending order.

Min Heap

Algorithm:

begin minHeap(list,n)

Arr[(i -1) / 2] returns its parent node.

Arr[(2 * i) + 1] returns its left child node.

Arr[(2 * i) + 2] returns its right child node.

end minheap

- **getMax():** It returns the root element of Max Heap. Time Complexity of this operation is O(1).
- **extractMax():** Removes the maximum element from MaxHeap. Time Complexity of this Operation is O(Log n) as this operation needs to maintain the heap property (by calling heapify()) after removing root.
- **insert():** Inserting a new key takes O(Log n) time. We add a new key at the end of the tree. If new key is smaller than its parent, then we don't need to do anything. Otherwise, we need to traverse up to fix the violated heap property.

Time Complexity:

Here there are two for loops, So time complexity is $O(n \log n)$.

Space Complexity:

Here there is one array of size n, So space complexity is $O(1)$.

Recurrence Relation:

T(n)= T(n-1) + O(log n)

Similarly,

T(n) = O(n) + n O(log n)

= **O(n log n)**

Implementation :

```
import sys

print("Niraj Italiya 190130107041")
print("5CEA2 Practical 3-Min Heap")

class MinHeap:

    def__init__(self, maxsize):
        self.maxsize = maxsize
        self.size = 0
        self.Heap = [0]*(self.maxsize + 1)
        self.Heap[0] = -1 * sys.maxsize
        self.FRONT = 1
```

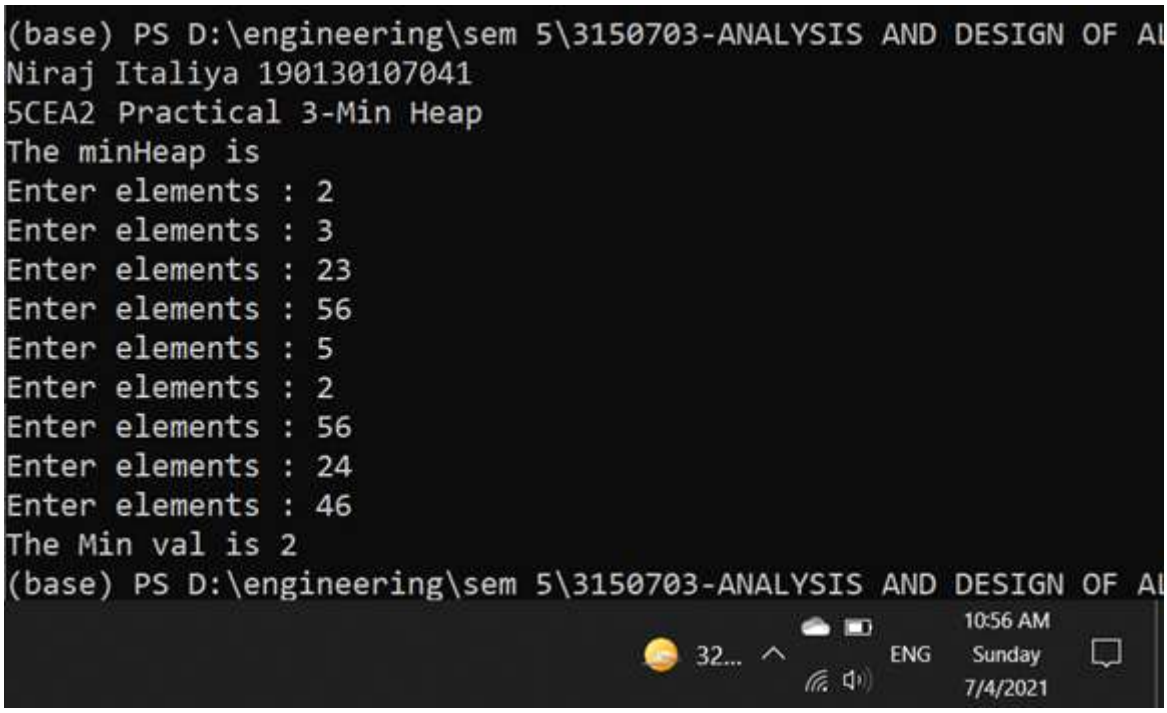
```
    def parent(self, pos):
        return pos//2

    def leftChild(self, pos):
        return 2 * pos

    def rightChild(self, pos):
        return (2 * pos) + 1

    def isLeaf(self, pos):
        if pos >= (self.size//2) and pos <= self.size:
            return True
        return False

    def swap(self, fpos, spos):
        self.Heap[fpos], self.Heap[spos] = self.Heap[spos],
self.Heap[fpos]

    def minHeapify(self, pos):

        if not self.isLeaf(pos):
            if (self.Heap[pos] > self.Heap[self.leftChild(pos)] or
                self.Heap[pos] > self.Heap[self.rightChild(pos)]):
```

```
            if self.Heap[self.leftChild(pos)] <
self.Heap[self.rightChild(pos)]:
                self.swap(pos, self.leftChild(pos))
                self.minHeapify(self.leftChild(pos))

            else:
                self.swap(pos, self.rightChild(pos))
                self.minHeapify(self.rightChild(pos))

    def insert(self, element):
        if self.size >= self.maxsize :
            return
        self.size+= 1
        self.Heap[self.size] = element

        current = self.size

        while self.Heap[current] <
self.Heap[self.parent(current)]:
            self.swap(current, self.parent(current))
            current = self.parent(current)
```

```
 def Print(self):
        for i in range(1, (self.size//2)+1):
            print(" PARENT : "+ str(self.Heap[i])+" LEFT CHILD : "+
                                str(self.Heap[2 * i])+" RIGHT CHILD : "+
                                str(self.Heap[2 * i + 1]))

    def minHeap(self):

        for pos in range(self.size//2, 0, -1):
            self.minHeapify(pos)

    def remove(self):

        popped = self.Heap[self.FRONT]
        self.Heap[self.FRONT] = self.Heap[self.size]
        self.size-= 1
        self.minHeapify(self.FRONT)
        return popped


if__name__== "__main__":

    print('The minHeap is ')
    minHeap = MinHeap(15)
```

```
for i in range(0, 9):
    x = int(input("Enter elements : "))
    minHeap.insert(x)


minHeap.minHeap()

print("The Min val is " + str(minHeap.remove()))
```

Output:



Conclusion:

- To sort a list of min heap sort it will take n iteration.
- After each the heaviest element is on the rightmost.
- Worst case would be when array is in descending order.

What have you learned from this experiment?

(Tick ✓ the appropriate)	Learning of	Write an appropriate answer in one or two words.
	Theoretical Concept	
✓	Problem Solving	Fine with it.
✓	Coding Style	Learn new Concept.
	Testing	
	Way to answer a question in exam	

Practical 4

Implementation and Time analysis factorial program using iterative and recursive method.

Algorithm:

Using Iteration:

```
factorial(number)
    FOR value = 1 to number
        factorial = factorial * value
        END FOR
    Return factorial
end
```

Using Recursion:

```
factorial(number)
    IF number == 0 or 1
        Return 1
    Else
        Return n * factorial(number-1)
end
```

Recurrence Relation:

$T(n)=T(n-1)+T(n-2)+\Theta(1)$

Similarly,

$T(n-1)=T(n-2)+n-1$

So on till,

$= T(1) + (2+3+\ldots+n)$

$=T(0) + (1+2+3+\ldots+n)$

$= T(0) + n*(n+1)/2$

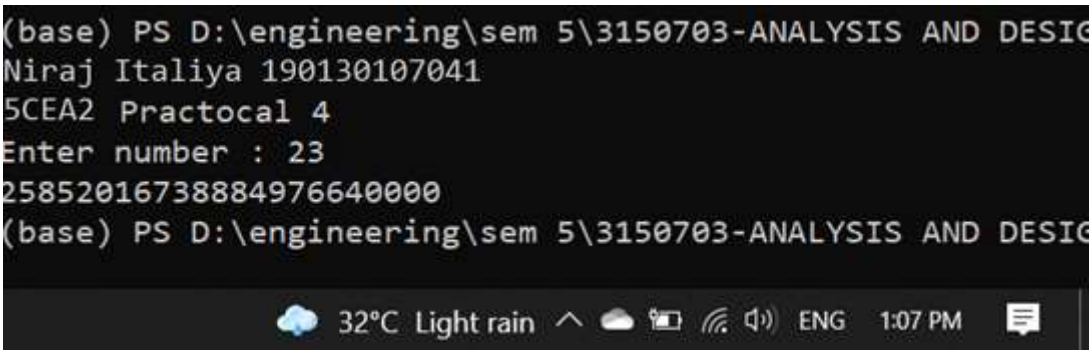
$= \Theta(n)$

Implementation using Recursion :

```
print(“Niraj Italiya 190130107041”)
print("Sem 5 - CE B2 Practocal 4")


def factorial(number):
    if number < 0:
        print('Invalid entry! Cannot find factorial of a
negative number')
        return -1
    if number == 1 or number == 0:
        return 1
```

```
else:
    return number * factorial(number - 1)
a = int(input("Enter number : "))
result = factorial(a)
print(result)
```

Output:



Recurrence Relation:

$T(n)=T(n-1)+T(n-2)+\Theta(1)$

Similarly,

$T(n-1)=T(n-2)+n-1$

So on till,

$= T(1) + (2+3+\ldots+n)$

$=T(0) + (1+2+3+\ldots+n)$

$= T(0) + n*(n+1)/2$

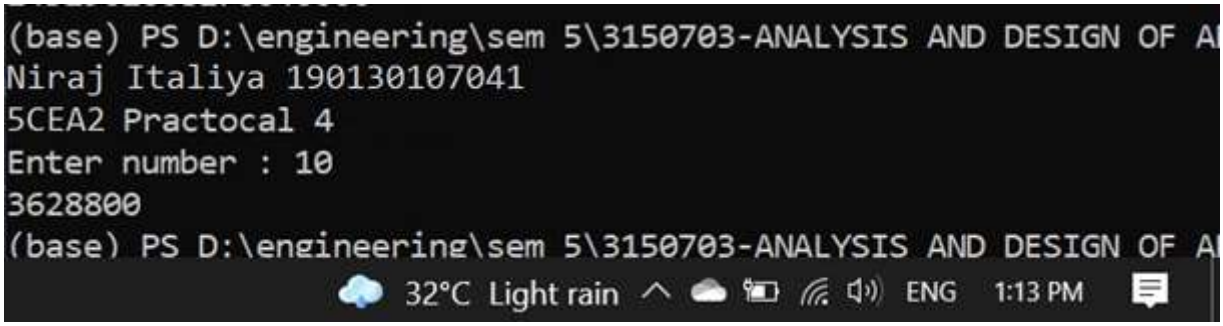
$= \Theta(n)$

Implementation using Iteration :

```
print("Niraj Italiya 190130107041")
print("Sem 5 - CE B2 Practocal 4")


def factorial_without_recursion(number):
    if number < 0:
        print('Invalid entry! Cannot find factorial of a
negative number')
        return -1
    fact = 1
    while(number > 0):
        fact = fact * number
        number = number - 1
```

```
        return fact
a = int(input("Enter number : "))
result = factorial(a)
print(result)
```

Output:



What have you learned from this experiment?

(Tick ✓ the appropriate)	Learning of	Write an appropriate answer in one or two words.
✔	Theoretical Concept	Got it Very Well
✔	Problem Solving	Logic building.
✔	Coding Style	Learn 2 method to count factorial.
	Testing	
	Way to answer a question in exam	

Practical 5

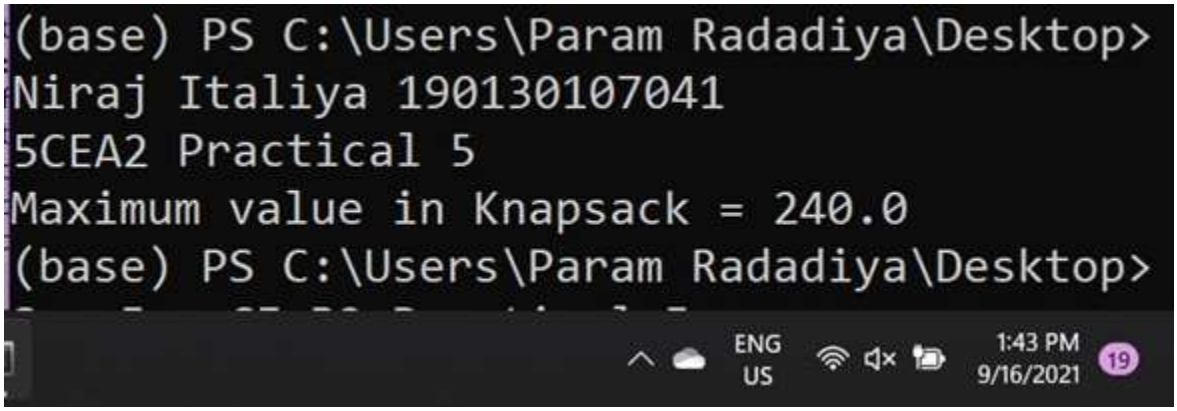
Implement Knapsak Problem using Greedy Method.

Implementation :

```
print(“Niraj Italiya 190130107041”)
print("Sem 5 - CE B2 Practical 5“)
class ItemValue:
        def__init__(self, wt, val, ind):
        self.wt = wt
        self.val = val
        self.ind = ind
        self.cost = val // wt

    def__lt__(self, other):
        return self.cost < other.cost

class FractionalKnapSack:
 def getMaxValue(wt, val, capacity):
        iVal = []
        for i in range(len(wt)):
            iVal.append(ItemValue(wt[i], val[i], i))
         iVal.sort(reverse = True)
```

```
        totalValue = 0
        for i in iVal:
            curWt = int(i.wt)
            curVal = int(i.val)
            if capacity - curWt >= 0:
                capacity -= curWt
                totalValue += curVal
            else:
                fraction = capacity / curWt
                totalValue += curVal * fraction
                capacity = int(capacity - (curWt * fraction))
                break
    return totalValue


if__name__== "__main__":
    wt = [10, 40, 20, 30]
    val = [60, 40, 100, 120]
    capacity = 50
maxValue = FractionalKnapSack.getMaxValue(wt, val,
capacity)
    print("Maximum value in Knapsack =", maxValue)
```

Output:


(base) PS C:\Users\Param Radadiya\Desktop>
Niraj Italiya 190130107041
5CEA2 Practical 5
Maximum value in Knapsack = 240.0
(base) PS C:\Users\Param Radadiya\Desktop>
ENG
US
1:43 PM
9/16/2021
19

What have you learned from this experiment?

(Tick ✓ the appropriate)	Learning of	Write an appropriate answer in one or two words.
✓	Theoretical Concept	New concept
✓	Problem Solving	Got more clarity.
✓	Coding Style	Learn new method to write program.
	Testing	
	Way to answer a question in exam	

Practical 6

Implementation the chain matrix multiplication using dynamic programming.

Implementation :

```
import sys
import time
import datetime
current_time = datetime.datetime.now()
initial = time.time()
dp = [[-1 for i in range(100)] for j in range(100)]
def matrixChainMemoised(p, i, j):
    if (i == j):
        return 0
    if (dp[i][j] != -1):
        return dp[i][j]
    dp[i][j] = sys.maxsize

    for k in range(i, j):
        dp[i][j] = min(dp[i][j],
        matrixChainMemoised(p, i, k) + matrixChainMemoised(p,
k + 1, j) + p[i - 1] * p[k] * p[j])
    return dp[i][j]
def MatrixChainOrder(p, n):
 i = 1
 j = n - 1
 return matrixChainMemoised(p, i, j)
```

```
print("Niraj Italiya  Practical- 6
\nDate/Time:",current_time)
arr = [11, 12, 13, 14, 15]
print("Array is",arr)
n = len(arr)
print("Minimum number of multiplications is",
MatrixChainOrder(arr, n))
```

Output:

```
Date/Time: 2021-08-09 13:55:44.887405
Array is [11, 12, 13, 14, 15]
Minimum number of multiplications is 6028
```

What have you learned from this experiment?

(Tick ✓ the appropriate)	Learning of	Write an appropriate answer in one or two words.
✓	Theoretical Concept	New concept
✓	Problem Solving	Learnt dynamic programming concept.
✓	Coding Style	It's new method to write program.
	Testing	
	Way to answer a question in exam	

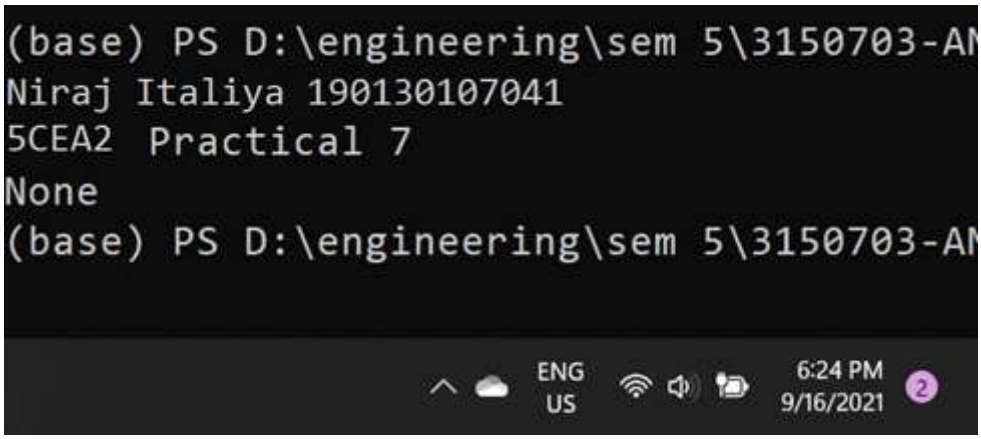
Practical 7

Implementation of making a change problem using dynamic programming

Implementation :

```
print("Niraj Italiya 190130107041")
print("5CEA2 Practical 7")

def count(S, m, n ):
 if (n == 0):
        return 1
 if (n < 0):
        return 0;
 if (m <=0 and n >= 1):
        return 0
        return count( S, m - 1, n ) + count( S, m, n-S[m-1] );
arr = [1, 2, 3]
m = len(arr)
print(count(arr, m, 4))
```

Output:



What have you learned from this experiment?

(Tick ✓ the appropriate)	Learning of	Write an appropriate answer in one or two words.
✓	Theoretical Concept	New concept
✓	Problem Solving	Learnt dynamic programming concept.
✓	Coding Style	It's new method to write program.
	Testing	
	Way to answer a question in exam	

Practical 8

Implementation of a knapsack problem using greedy algorithm.

Implementation :

```
import time
import datetime
current_time = datetime.datetime.now()
initial = time.time()
# Python3 program to solve fractional
# Knapsack Problem
class ItemValue:
    """Item Value DataClass"""
    def __init__(self, wt, val, ind):
        self.wt  =  wt
        self.val  =  val
        self.ind = ind
        self.cost  =  val  //  wt
    def __lt__(self,  other):
        return self.cost < other.cost
# Greedy Approach
class FractionalKnapSack:
    """Time Complexity O(n log n)"""
    @staticmethod
    def getMaxValue(wt, val, capacity):
        """function to get maximum value """
        iVal = []
        for i in range(len(wt)):
            iVal.append(ItemValue(wt[i], val[i], i))
```

```
        iVal.sort(reverse=True)
        totalValue = 0
        for i in iVal:
            curWt = int(i.wt)
            curVal = int(i.val)
            if capacity - curWt >= 0:
                capacity -= curWt
                totalValue += curVal
            else:
                fraction = capacity / curWt
                totalValue += curVal * fraction
                capacity = int(capacity - (curWt * fraction))
                break
        return totalValue
# Driver Code
if__name___== "__main__":
    wt = [10, 40, 20, 30]
    val = [60, 40, 100, 120]
    capacity = 50
 # Function call
    maxValue = FractionalKnapSack.getMaxValue(wt, val,
capacity)
print(current_time)
print("Maximum value in Knapsack =", maxValue)
```

Output:

```
  2021-08-09 14:26:05.564699
Maximum value in Knapsack = 240.0
PS C:\Users\Param Radadiya>
```

What have you learned from this experiment?

(Tick ✓ the appropriate)	Learning of	Write an appropriate answer in one or two words.
✔	Theoretical Concept	Got the concepts of knapsack problem.
✔	Problem Solving	Learnt various greedy methods.
✔	Coding Style	Definitely improved.
	Testing	
	Way to answer a question in exam	

Practical 9

Implementation of Graph and Searching (DFS and BFS).

Implementation :

DFS (Depth First Search)

```
print("Niraj Italiya  1901301070441")
print("5CEA2 Practical 9")

import time
initial = time.time()
graph = {
 'A' : ['B','C'],
 'B' : ['D', 'E'],
 'C' : ['F'],
 'D' : [],
 'E' : ['F'],
 'F' : []
}
visited = set()
def dfs(visited, graph, node):
 if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
                dfs(visited, graph, neighbour)

dfs(visited, graph, 'A')
print("\n","Time=",time.time() - initial)
```

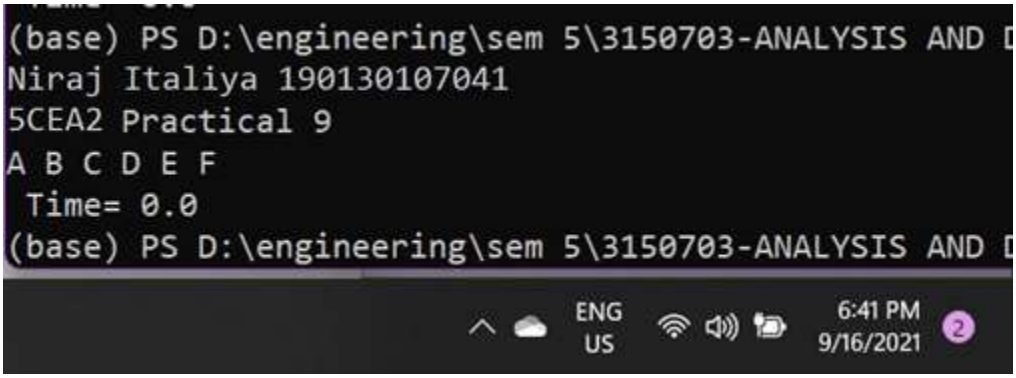
Output:

BFS (Breadth First Search)

```
print("Niraj Italiya  190130107041")
print("5CEA2 Practical 9")

import time
initial = time.time()
graph = {
 'A' : ['B','C'],
 'B' : ['D', 'E'],
 'C' : ['F'],
 'D' : [],
 'E' : ['F'],
 'F' : []
}
visited = []
```

```
queue = []
def bfs(visited, graph, node):
 visited.append(node)
 queue.append(node)
 while queue:
   s = queue.pop(0)
   print (s, end = " ")
   for neighbour in graph[s]:
     if neighbour not in visited:
       visited.append(neighbour)
       queue.append(neighbour)

bfs(visited, graph, 'A')
print("\n","Time=",time.time() - initial)
```

Output:



What have you learned from this experiment?

(Tick ✓ the appropriate)	Learning of	Write an appropriate answer in one or two words.
✓	Theoretical Concept	Got all the concepts.
✓	Problem Solving	Learnt searching in various type data.
	Coding Style	
	Testing	
	Way to answer a question in exam	

Practical 10

Implementation prism's Algorithm for minimum spanning tree.

```
import datetime
current_time = datetime.datetime.now()

print("Niraj Italiya  Practical 10\n", current
_time)

INF = 9999999
V = 5
G = [[0, 9, 75, 0, 0],
 [9, 0, 95, 19, 42],
 [75, 95, 0, 51, 66],
 [0, 19, 51, 0, 31],
 [0, 42, 66, 31, 0]]
selected = [0, 0, 0, 0, 0]
# set number of edge to 0
no_edge = 0
selected[0] = True
print("Edge : Weight\n")
while (no_edge < V - 1):
    minimum = INF
    x = 0
    y = 0
```

```
for i in range(V):
        if selected[i]:
            for j in range(V):
                if ((not selected[j]) and G[i][j]):
                    if minimum > G[i][j]:
                        minimum = G[i][j]
                        x = i
                        y = j
    print(str(x) + "-" + str(y) + ":" + str(G[x][y]))
    selected[y] = True
    no_edge += 1
```

Output:

```
 2021-08-09 14:33:24.035935
Edge : Weight

0-1:9
1-3:19
3-4:31
3-2:51
PS C:\Users\Param Radadiya>
```

What have you learned from this experiment?

(Tick ✓ the appropriate)	Learning of	Write an appropriate answer in one or two words.
✓	Theoretical Concept	Learnt new concepts.
✓	Problem Solving	Definitely increased.
	Coding Style	
	Testing	
	Way to answer a question in exam	

Practical 11

lmplementation Kruskal's Algorithm for minimum spanning tree.

```
import datetime
current_time = datetime.datetime.now()
print("Niraj Italiya 190130107041 Practical 11\n", current
_time)

class Graph:
    def__init__(self, vertices):
        self.V = vertices
        self.graph = []
    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])
    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])
    def apply_union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
            rank[xroot] += 1
    def kruskal_algo(self):
```

```
        result = []
        i, e = 0, 0
        self.graph = sorted(self.graph, key=lambda item: item[2]
)
        parent = []
        rank = []
        for node in range(self.V):
            parent.append(node)
            rank.append(0)
        while e < self.V - 1:
            u, v, w = self.graph[i]
            i = i + 1
            x = self.find(parent, u)
            y = self.find(parent, v)
            if x != y:
                e = e + 1
                result.append([u,          v,          w])
                self.apply_union(parent, rank, x, y)
        for u, v, weight in result:
            print("%d - %d: %d" % (u, v, weight))

g = Graph(6)
g.add_edge(0, 1, 4)
g.add_edge(0, 2, 4)
g.add_edge(1, 2, 2)
g.add_edge(1, 0, 4)
g.add_edge(2, 0, 4)
g.add_edge(2, 1, 2)
g.add_edge(2, 3, 3)
g.add_edge(2, 5, 2)
g.add_edge(2, 4, 4)
```

```
g.add_edge(3, 2, 3)
g.add_edge(3, 4, 3)
g.add_edge(4, 2, 4)
g.add_edge(4, 3, 3)
g.add_edge(5, 2, 2)
g.add_edge(5, 4, 3)
g.kruskal_algo()
```

Output:

```
 2021-08-09 14:44:13.520643
1 - 2: 2
2 - 5: 2
2 - 3: 3
3 - 4: 3
0 - 1: 4
```

What have you learned from this experiment?

(Tick ✓ the appropriate)	Learning of	Write an appropriate answer in one or two words.
✔	Theoretical Concept	Very Well
	Problem Solving	
✔	Coding Style	Learnt new Concept.
	Testing	
✔	Way to answer a question in exam	Enough

Practical 12

Implementat lCS Problem.

```
import datetime
current_time = datetime.datetime.now()
print("Niraj Italiya  190130107041 Practical 12\n", current
_time)

def lcs_algo(S1, S2, m, n):
    L = [[0 for x in range(n + 1)] for x in range(m + 1)]
    for i in range(m + 1):
        for j in range(n + 1):
            if i == 0 or j == 0:
                L[i][j] = 0
            elif S1[i - 1] == S2[j - 1]:
                L[i][j] = L[i - 1][j - 1] + 1
            else:
                L[i][j] = max(L[i - 1][j], L[i][j - 1])
    index = L[m][n]
    lcs_algo = [""] * (index + 1)
    lcs_algo[index] = ""
    i = m
    j = n
    while i > 0 and j > 0:
        if S1[i - 1] == S2[j - 1]:
            lcs_algo[index - 1] = S1[i - 1]
            i -= 1
```

```
            j -= 1
            index -= 1
        elif L[i - 1][j] > L[i][j - 1]:
            i -= 1
        else:
            j -= 1
        print("S1 : " + S1 + "\nS2 : " + S2)
        print("LCS: " + "".join(lcs_algo))

S1 = "ACADB"
S2 = "CBDA"
m = len(S1)
n = len(S2)
lcs_algo(S1, S2, m, n)
```

Output:

```
 2021-08-09 14:49:37.836883
S1 : ACADB
S2 : CBDA
LCS:
S1 : ACADB
S2 : CBDA
LCS:
S1 : ACADB
S2 : CBDA
LCS: B
S1 : ACADB
S2 : CBDA
LCS: B
S1 : ACADB
S2 : CBDA
LCS: B
S1 : ACADB
S2 : CBDA
LCS: CB
```

What have you learned from this experiment?

(Tick ✓ the appropriate)	Learning of	Write an appropriate answer in one or two words.
✓	Theoretical Concept	Totally new concept.
	Problem Solving	
✓	Coding Style	Gogd dome idea.
	Testing	
	Way to answer a question in exam	

1901301047041

Assignment :- ADA

Tutorial :- 1

(1) What is an algorithm ? Explain various properties of any algo

→ Algorithm is a step by step procedure which define a set of instruction to be executed in a certain order to get the desired output. Algorithmes are generally created independent of underlying languages

↳ charactics of a Algorithm

unambiguous :- Algorithm should be clear and unambiguous fetch of its steps and other iiri ois should be clear and must lead to only one meaing

Input :- An Algorithm should have or more well defined inputs

output :- An Algorithm should have for more well defined outputs and should match the desired output

finiteness :- Algorithms must terminate after a finite number of steps.

feasibility :- should be feasible with avaiable resources.

Independent:- An Algorithm should have step-by-step directions which should be independent of any programm code.

What do you mean by Analysis of algorithms.

→ Analysis of algo is the determinat of the amount of time and space resources required to execute it.

→ Usually, the efficency or running time of an algo is stated as function relating the input length to number of steps, know as time complexity or volumes memory known as space complexity

(3) Explain various type of Algorithm with example

1) Recursive Algorithm

→ problems such as the tower of Hanoi or DFS a graph can be easily solved by using these algo.

→ It's algo that calls itself recursively until problems is solved.

Eg. Here is code that finds functional using a recursive algo

```
fact (y)
if y is 0
return 1
return (y* fact(y-1))
```

(2) Divide and conquer Algo.

→ in divide and conquer algorithms devide the Algo. in two parts, the first part divide the problem on hand in to smaller subproblems of the same type, then second part the smaller problem are solved and add together, to produce the problem's final solution merge sorting and quick sorting can be done with devdie and conquer Algo.

Ex: mergesorting ([arr], i, r) if r>1

1) find the mid point to devide the given array in to halves:
middle m = (l+r)/2

2) call merge sorting for the first half
call merge sorting (arr, l, m)

3) call merge sorting for second half
call merge sorting (arr, m+1, r)

4) merge the halves sorted in step 2 and 3
call merge (arr, l, m, r)

3) Dynmic programming Algo
→ this Algo solves complex Algo problems by breaking them in to multiple simple subproblems and then it solve each of them once and then stores them for further use.

Ex: fibonacci sequence

fibonacci (N) = 0 (for (n=0))
= 0 (for (n=1))
= fibonacci (N-1) + finacchi (N-2). (for n>1)

4) Greedy Algo

→ These Algo are used for solving optimization problems.

→ Huffman coding and Dijkstries Algo are two prime exmple where the greedy algo is used.

5) Brute force Algo

→ A Brute force of Algo blindly iteracter all posible solutions to search one or more than one solution that may solve a function

Ex: sequantial search

algorithm s-search (A[0,n], X)
A(n) ← X
i ← 0
if i<n return
else return n-1

6) Backtracking Algo

→ Backtracting Algo solves sub problem and if it the problem it does that last step and starts again to find the solution to the problem.

Define: subset empty set nullsety power set equal set.

1) subset

→ A set A is subset of another set B if all element of set A or element of the set B

$A \subset B$

2) empty set.

→ empty set is the unique set having no elements.

→ ϕ

3) null string

→ The empty string is the special case where the sequence has length zero

4) Power Set

→ A power set is defined as the set or group of all subset for any given set.

5) Equal set

→ If two set have the same and equal element is called equale set

5) If $A = \{1, 2, 3, 4, 5\}$

$B = \{1, 3, 5, 7, 9\}$

$U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

(i) $A \cup B = \{1, 2, 3, 4, 5, 7, 9\}$

(ii) $A \cap B = \{1, 2, 3, 4, 5, 7, 9\}$

(iii) $A - B = \{2, 4\}$

(iv) $A' = \{1, 2, 3, 4, 5\}$

(v) $A' \cap B =$

{(1,1), (1,3), (1,5), (1,7), (1,9),
(2,1)), (2,3), (2,5), (2,7), (2,9),
(3,1), (3,3), (3,3), (3,7), (3,7),
(4,1)), (4,3), (4,5), (4,7), (4,9),
(5,1)) , 5(5,3), (5,5), (5,7), (5,9) }

(6) Arrange following of growth in increasing order.

2^n, $n \log n$, n^2, 1, n, $\log n$, $n!$, n^3.

Ans-) 1, $\log n$, n, 2^n, $2n \log n$, n^2, n^3, $n!$

(7) Define :- Time completity and space completity worst case, best case, Average case

→ Time complexity:

:- Time complexity is the amount of time take by an algorithm to run.

→ space complexity:

→ space complexy is the amount of space taken taken by an algorithm to run

→ worst case Algo.

→ in the worst case analysis we calculate the upper bound on the running time of an algo. we must know the case that causes maximum number of operations to be executed for linear search.

→ The worst case happens when the element to be searched is not present in the array.

when x is not present, the search() function compares it with all elements of arr[] one by one. Therefore, the worst case time complexity of linear search would be O(n).

→ Average case analysis

→ In average case analysis we take all possible inputs and calculate computing time for all of the inputs sum all the calculated values and divide the sum by the total number of inputs.

→ we must know the distribution of case for the learner search problem. Let assume that all cases are uniformly distributed so we sum all the cases and divide the sum by (n+1)

→ Average case time

$$\frac{\sum_{i=1}^{n+1} O(i)}{n+1} = \frac{O[(n+1)*(n+1)/2]}{(n+1)}$$

$$= O(n)$$

→ Best case analysis

→ In Best case Analysis we calculate the lower bound on the running time of an algorithm we must know the case that causes when x is present is first location.

→ so time complexity is O(1)

8) what are asymptotic notations.

→ Asymptotic notations are mathematical tools to represent the time complexity of algorithm for asymptotic analysis

9) Define:- Big oh, Big omega theta

(1) Θ Notation.

: The theta notation bounds function from above and below so it defins exact asymptotic behaviour.

$c_1 g(n) <= f(n) <= c_2 g(n)$
for all $n \geq n_0$

(2) Big O Notation:

→ The Big O notation defince an upper bound of a algo it bounds a fun only from above.

$f(n) <= c\, g(n)$ for all $n >= n_0$

3) Ω Notation:

→ Ω Notation provides an asymptotic lower bound.

→ It is useful when we have lower bound on time complexity of an algorithm

$c\, g(n) \leq f(n)$ for all $n >= n_0$

Tutorial :- 2

(1) Analize following sorting method in best & worst case.

1) Selection sort

→ Algo.

```
For  i ← 1  to n-1 : do
     min j ← i;
     min x ← A[i]
     For  j ← j+1 to do
     If  A[j] < min x then
          min x ← A[j]
          A[min P] ← A[j]
          A[i] ← min x.
```

→ Best case (Assending order)

	0	1	2	3	4	5
Ex:-	10	20	30	40	50	60

↑ min = 10

$\frac{n(n-1)}{2} = O(n^2)$

→ worst case (Desending order)

Ex: $O(n^2)$

→ Averge :- $O(n^2)$

2) Bubble sort :-

→ code :-

```
for (i=0; i<n-1; i+1)
{ for (j=0; j<n; j++)
  {
    if (A[j] > A[j+1])
    {
```

```
temp = A[j]
A[j] = A[j+1];
A[j+1] = temp;
}
}
}
```

→ Best case :- Elements are already sorted

0	1	2	3	4
5	6	8	15	16

Passes $n-1$ comparisons in each pass

Total no of comparisons $(n-1)(n-1)$

$n^2 - 2n + 1 = O(n^2)$

→ Worst Elements are in reverse order also $(O(n^2))$

→ Average case :- $(n-1)(n-1) = O(n^2)$

(ii) Insertion sort

code :-

```
for (int i=0; i<n; i++)
{ temp = A[i];
  j = i-1;
  while (j>=0 && A[j] > temp)
  {
    A[j+1] = A[j]
    j--;
  }
  A[j+1] = temp
}
```

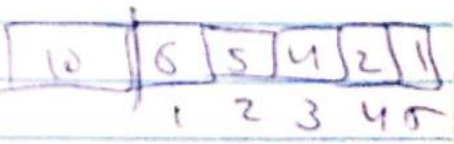
-) Best case (ARR sorted)

0	1	2	3	4	5	No of comparison
0	2	4	5	6	10	1
0	2	4	5	6	10	1
0	2	4	5	6	10	1
0	2	4	5	6	10	1
0	2	4	5	6	10	1
0	2	4	5	6	10	1 (N-1)

Best case (N) N=6

Worst case : (descending order) comparison

N=6



total no of comparison

$\therefore 1+3+4+5$

$\therefore 1+2+3+4+(N-1)$

$\therefore \frac{N(N-1)}{2}$ N^2-N

Worst case: N^2

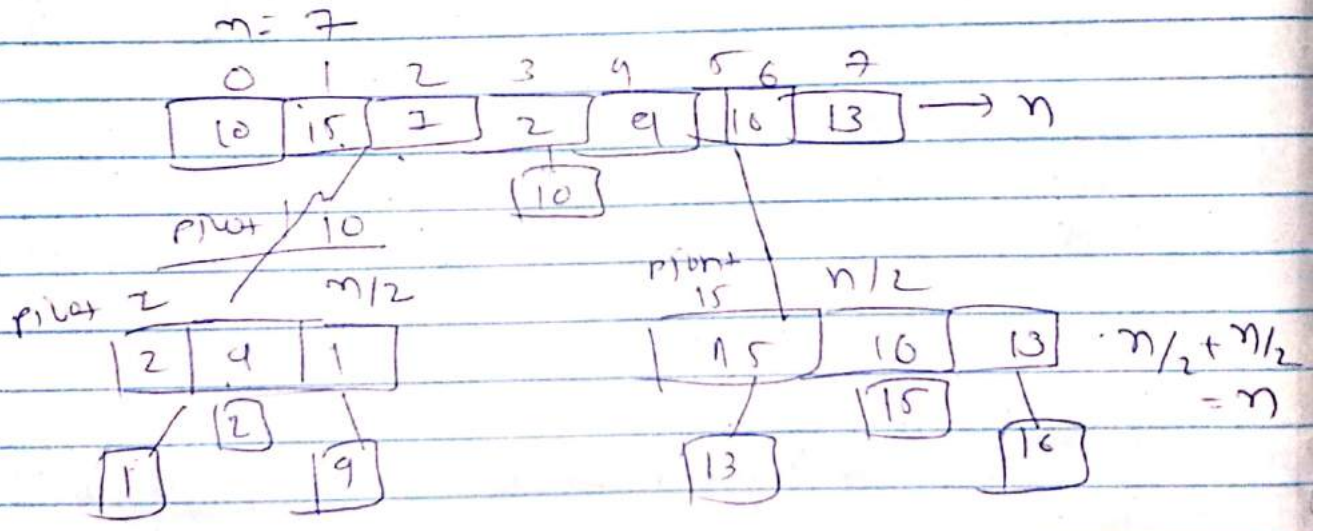
Average case:-

$O(n^2)$

(iv) Quick sort

```
{ if (l<h)
  { loc = partition (l, h);
    quick sort (l, loc-1);
    quick sort (loc+1, h);
  }
}
```

→ Best case



Time complexity of entire recursion tree

= No of levels * time taken for one level.

$n/2^k = 1$ $\quad n = 2^k$

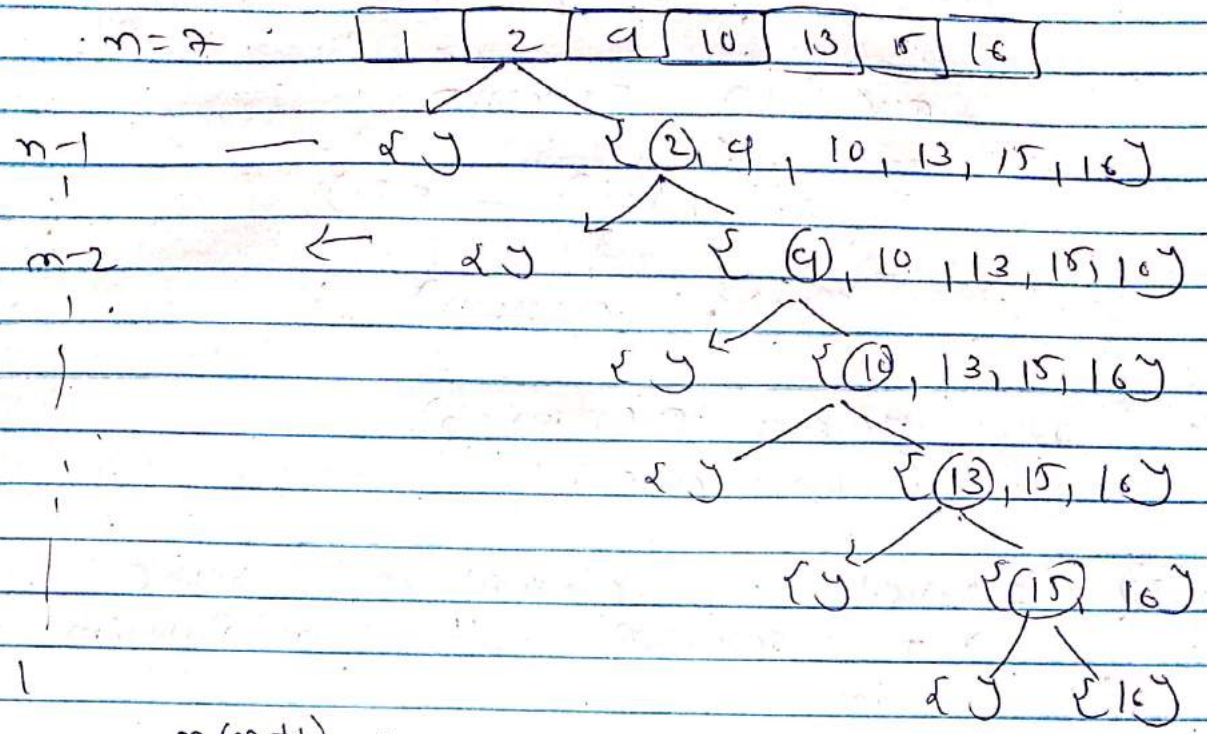
$\log n = k \log 2$

$k = \frac{n \cdot \log n}{2}$

Best case:-

$O(n * \log n)$

Worst case:-



$= \frac{n(n+1)}{2}$

$= n^2 + n$

$\neq O(n^2)$

(V) Heap sort

1) max Heapify
2) Build max Heap
3) Heap sort

1) time complexity of max Heapify

2) time taken to fix up the relationship among elements
A[i], A[left[i]], A[right[i]]

n = no of nodes

any subtree of that tree $\frac{2n}{3}$ nodes.

max $\frac{n}{3}$ elements

$$T(n) = T\left(\frac{2n}{3}\right) + \Theta(1)$$

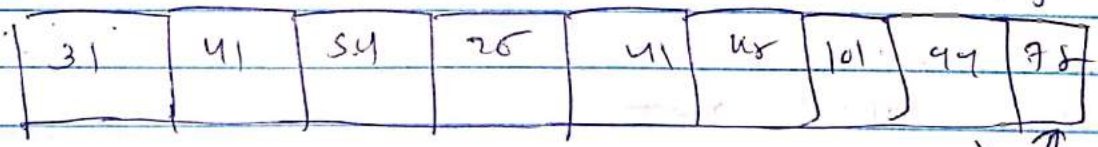
$$T(n) = \Theta(\log n)$$

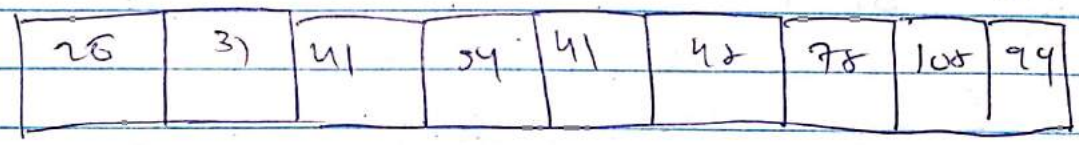
2) Heap sort $\Theta(n \log(n))$
(worst case)

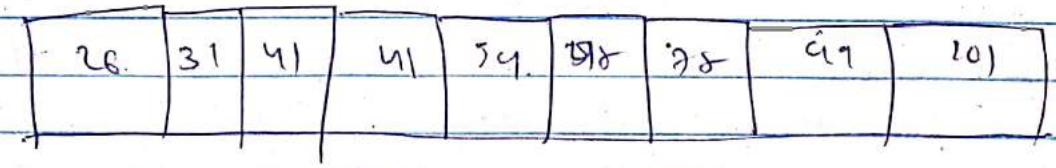
2) Illustrate the operation of bubble sort selection sort on following any.

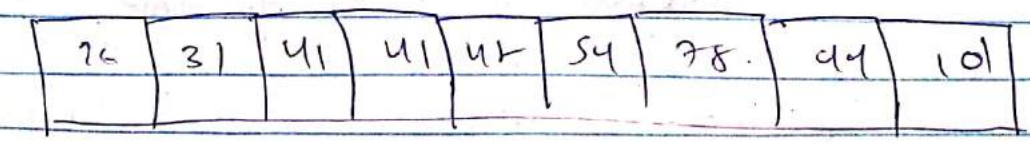
→ Bubble sort

D [31, 41, 54, 26, 41, 42, 101, 44, 78]

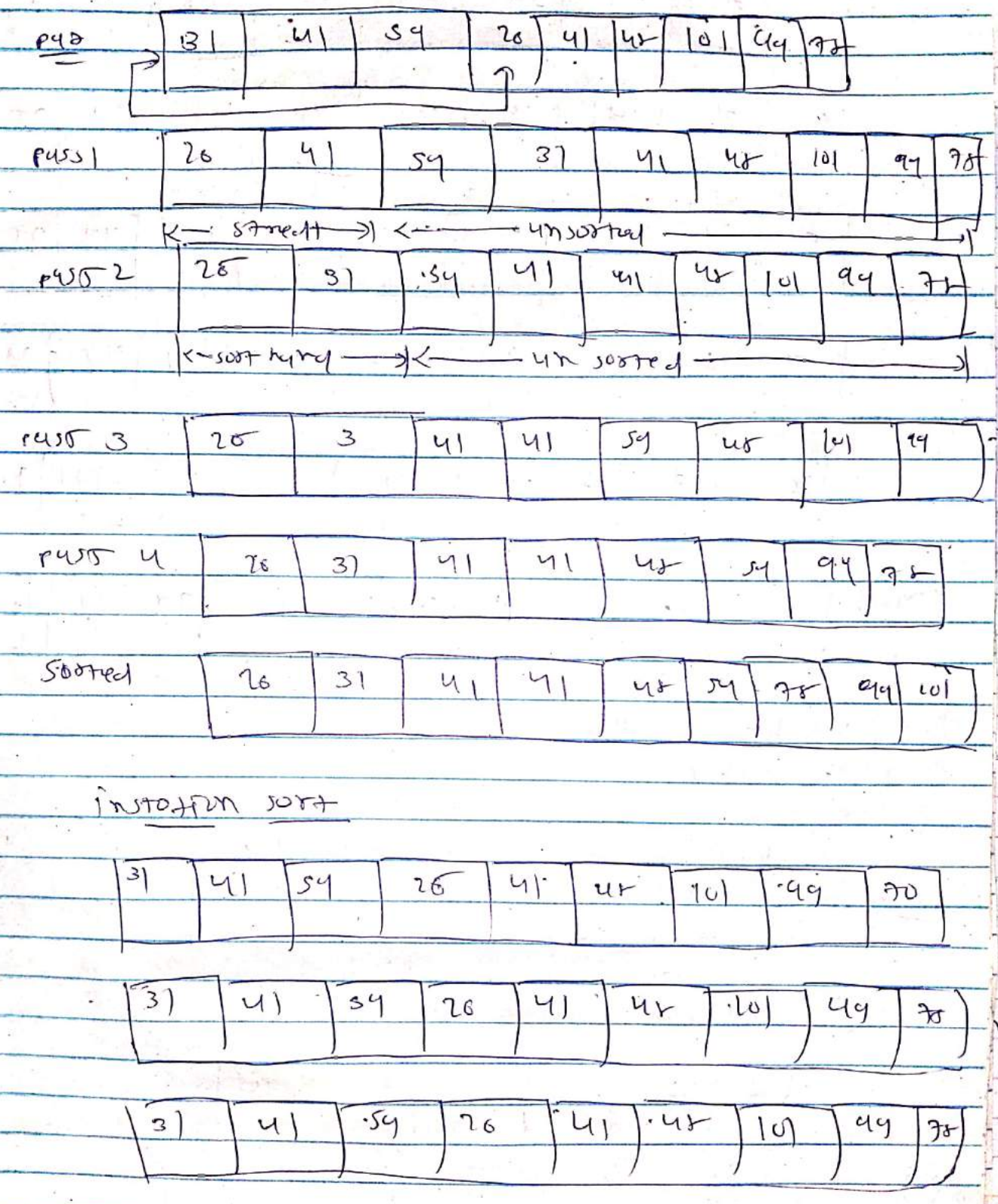








selection sort



insertion sort

26	31	41	54	41	52	10	99	75

26	31	41	54	41	52	10	99	75

26	31	41	41	52	54	10	99	75

26	31	41	41	52	54	10	99	75

26	31	41	41	52	54	99	10	75

sorted	26	31	41	41	52	54	99	10

(2) <1, 3, 5, 8, 9, 10>

(1) Bubble sort

	1	3	5	8	9	10
sorted	1	3	5	8	9	10

(2) selection sort

→	1	3	5	8	9	10

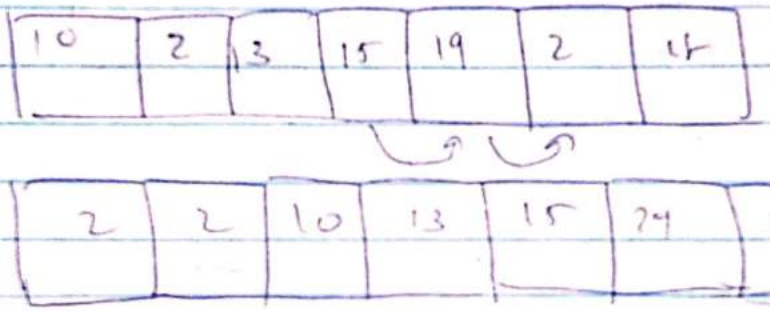
(3) Insertion sort

1	3	5	7	9	10

⇒ already sorted

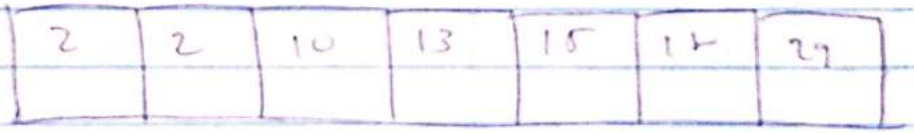
(3) <10, 2, 13, 15, 14, 2, 17>

(1) Bubble sort

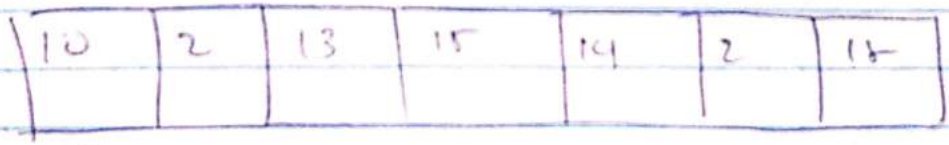


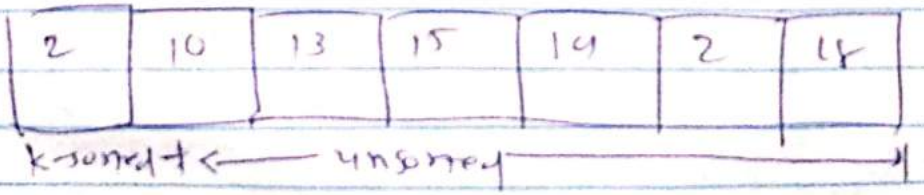
sorted

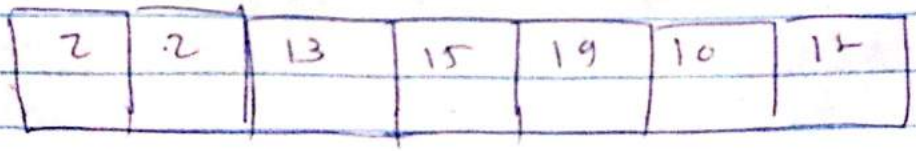
sorted

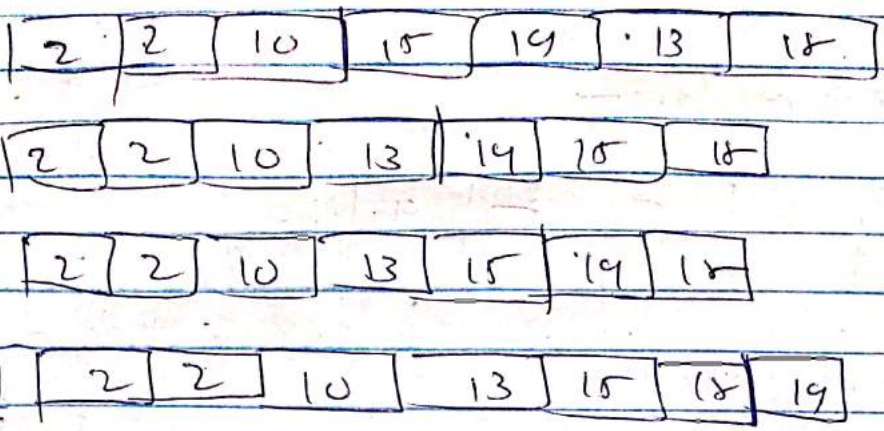


2) selection sort

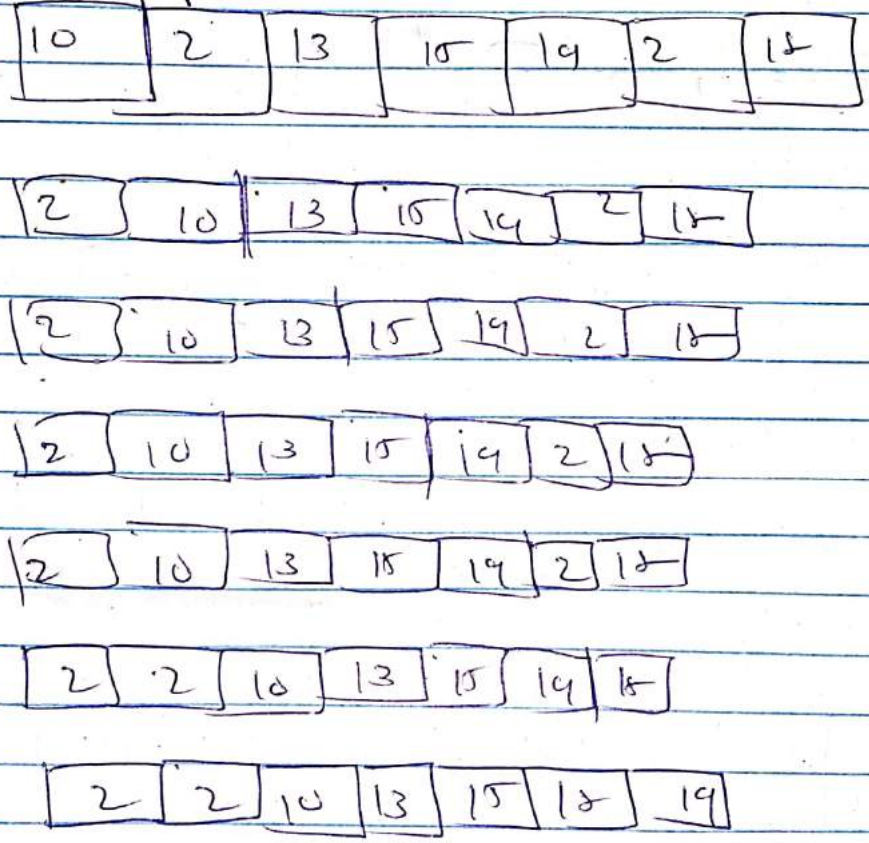








3) Insertion sort



→ Analize sequential search and binary search in best and worst case

→ Linear_search (a,n, item, loc)

Begin
for i = 0 to (n-1) by 1 do
if [a[i] = item) then
set loc = i
Exit
end if
end for
set loc = -1
End

Best case
→ The element being searched may be found at first position

→ In this case, the search terminates in success with just one comparison

→ Thus in best case, linear search algorithm takes O(1) operations.

Worst case:-
→ The element being searched may be present at the last position or not present at in the array at all

→ In the form case the search terminates in success with n comparisons

In the latter case the search terminates in failure with n comparisons.

→ Thus in worst case linear algo takes O(n) operations

→ Thus.

→ Time:- Complexity of linear search algorithm is O(n)

* Binary search Algo.

```
Begin
set beg = 0
set end = n-1
set mid = (beg + end)/2
while ((beg < end) and (a[mid] ≠ item))
                                    do
if (item < a[mid]) then
    set end = mid - 1
else
    set beg = mid + 1
    end if
set mid = (beg + end)/2;
end while
if (beg > end) then
    set loc = -1
else
    set loc = mid
    end if
End
```

→ Analysis

→ In each iteration or in each recursive call, the search get reduced to half of the array so for n elements in the array there are logn iterations recursive calls.

Thus we have

⇒ Time complexity of binary search is O(log n)

(4) compare Iterative and Recursive algo to find out fibnacci sences

→ Iterative

```
def functional(number):
    product = 1
    for i in range(number)
        product = product * (i+1)
    return product
```

→ functional [0, 1, 2]

i	product * (i+1)	product
0	1 * (0+1)	1
1	1 * (1+1)	2
2	2 * (2+1)	6

Recursive

```
def factorial (number):
    if number <=1:
        return 1:
    else.
        return number * factorial(number)
```

factorial (3) = 6

↓

3 * factorial (2)

↓

2 * factorial(1)

↓

1

(5) What do you mean by amortized analysis?

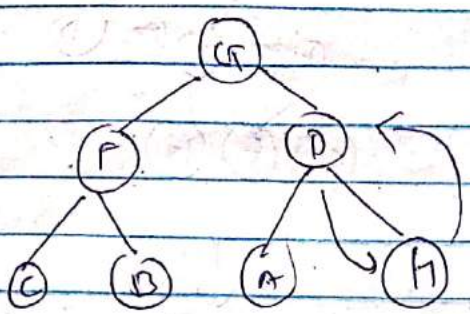
→ Amortized analysis is worst case analysis of a sequance of operation to obtain a tighter bound on the overall or average cost per operation in the sequence than is obtained by separatable analyzing each operation in the sequence

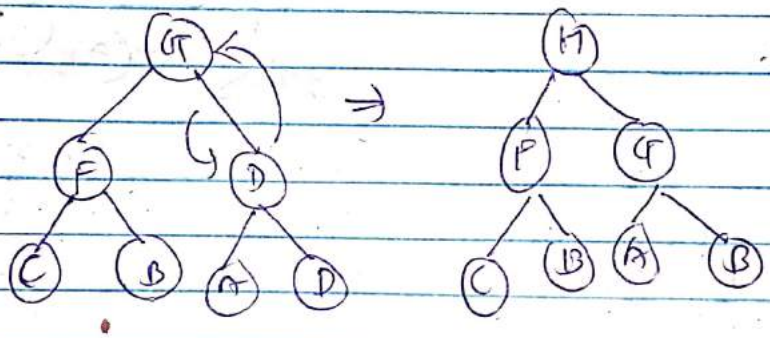
(6) Create min heap and max heap for following data

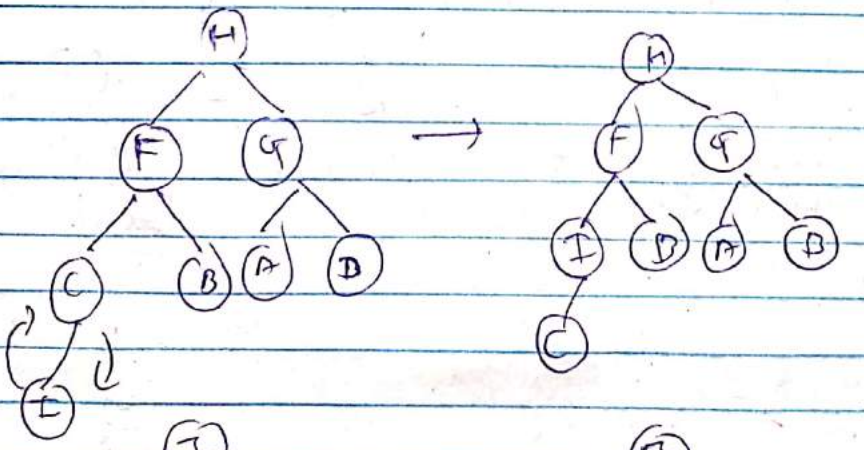
1) <4, E, D, C, B, H, I, J, K>

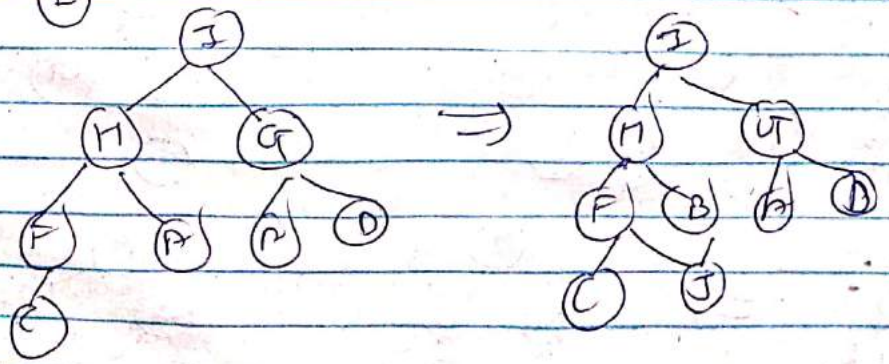
Max Heap

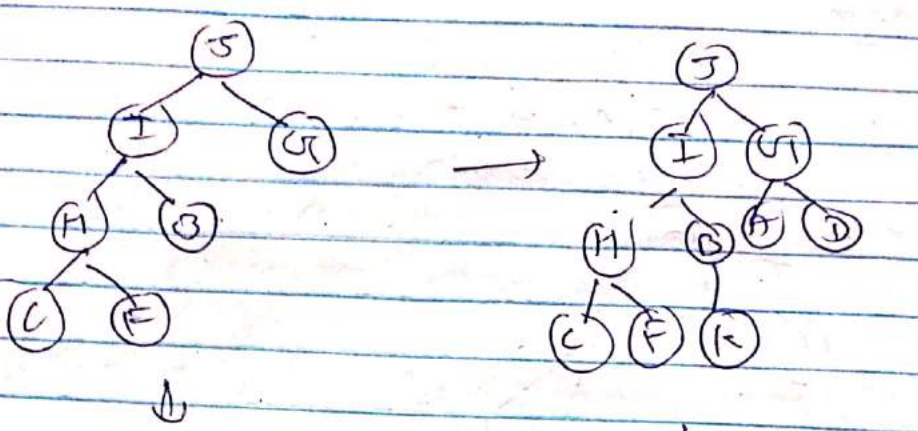
< G, F, D, C, B, A, H, I, J, K >







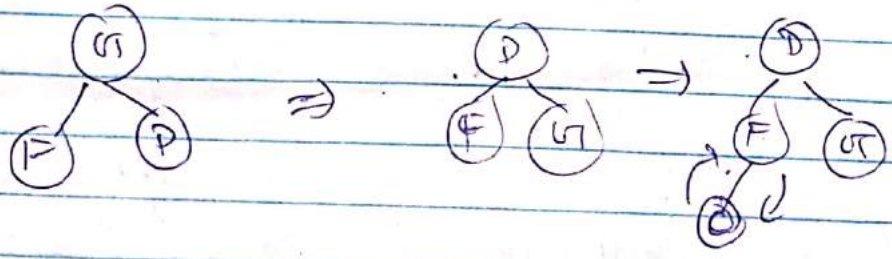


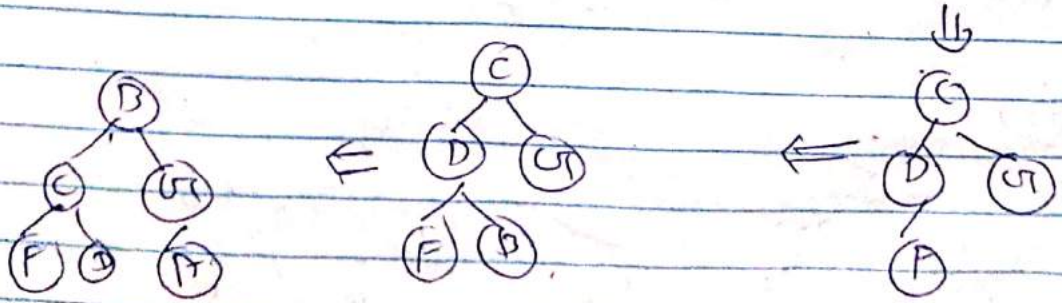


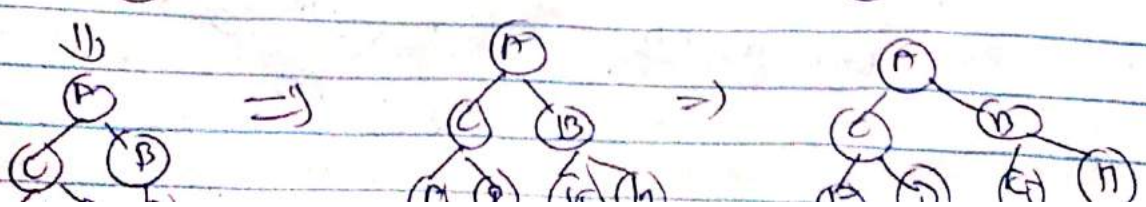
max Heap

min Heap

(1) < G, F, D, C, B, A, H, I, J, K >

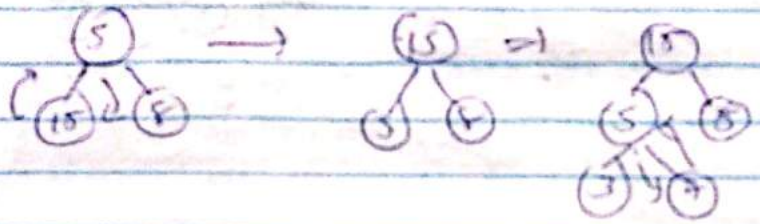


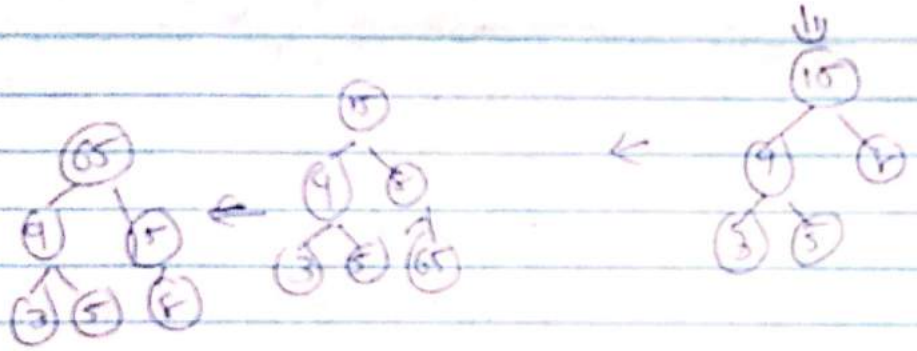




(2) <5, 15, 6, 3, 9, 65>

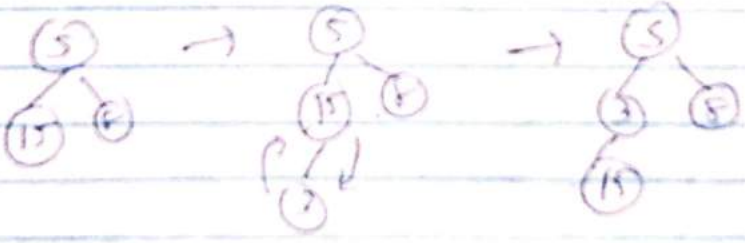
max HEP





max HEAP

min HEAP

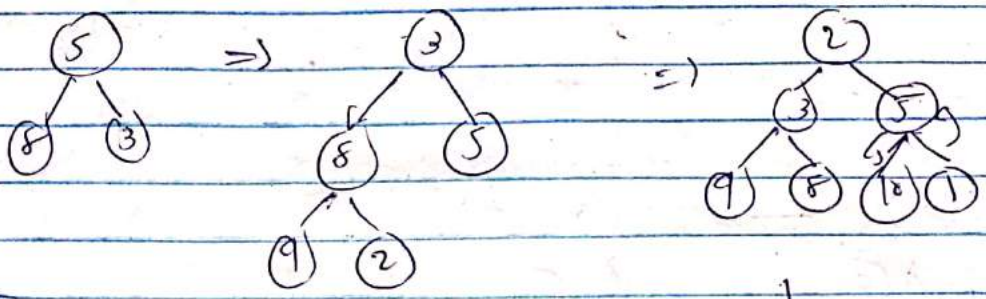


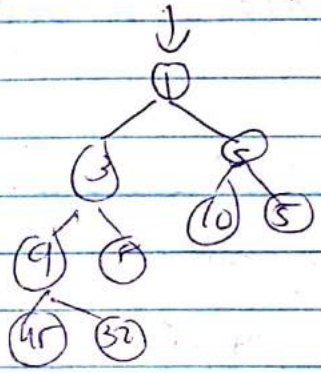
← min HEAP

(3) Sort following array using HEAP sort

(1) <5, 15, 3, 9, 2, 10, 1, 45, 32>

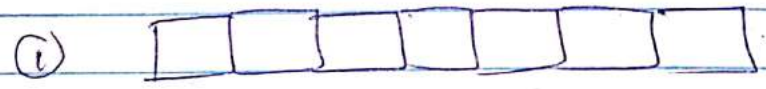
MAX HEAP

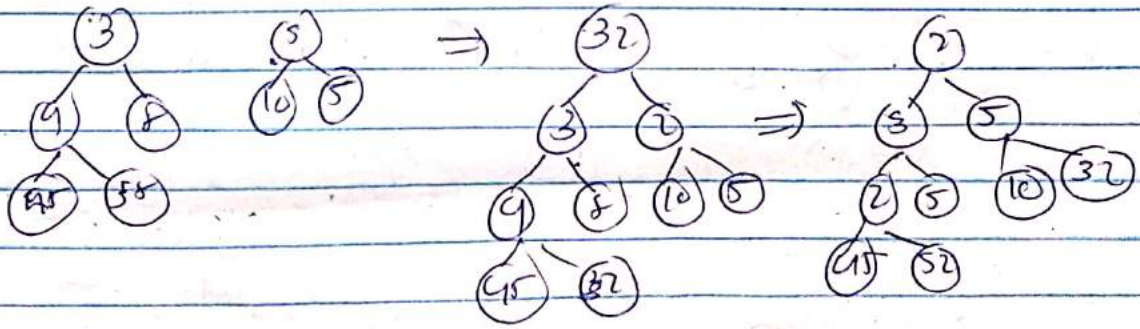


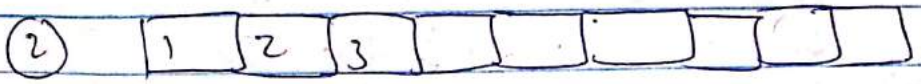


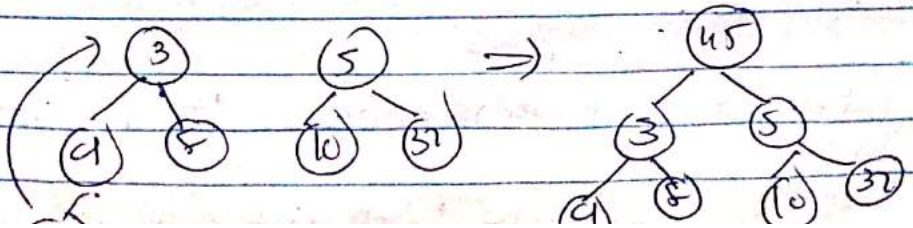
min heap

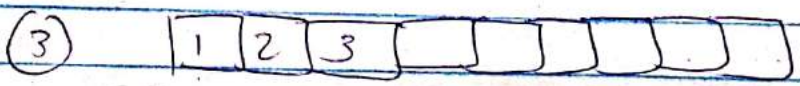
remove root from min heap and put in to array

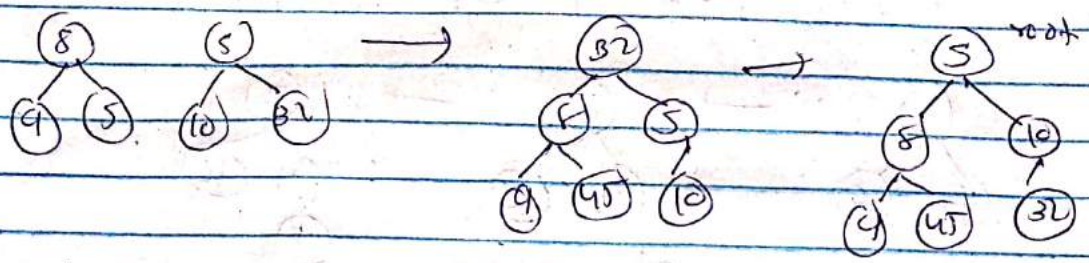


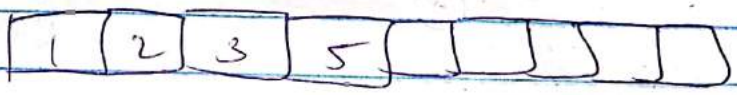


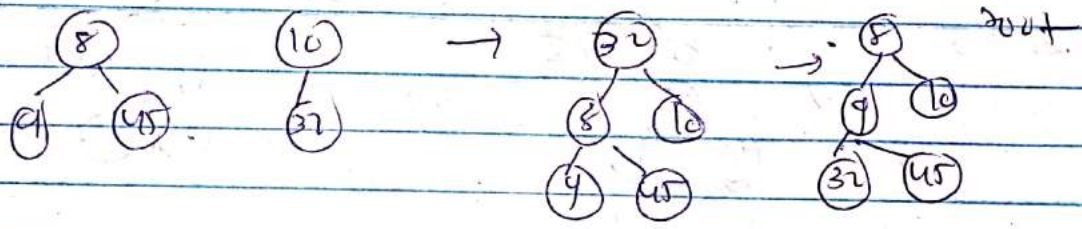


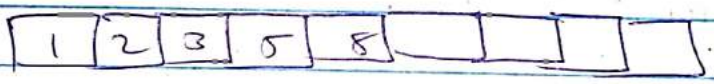


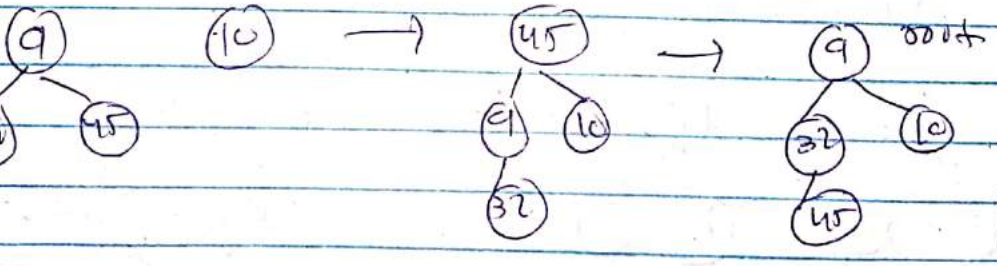
(3) 



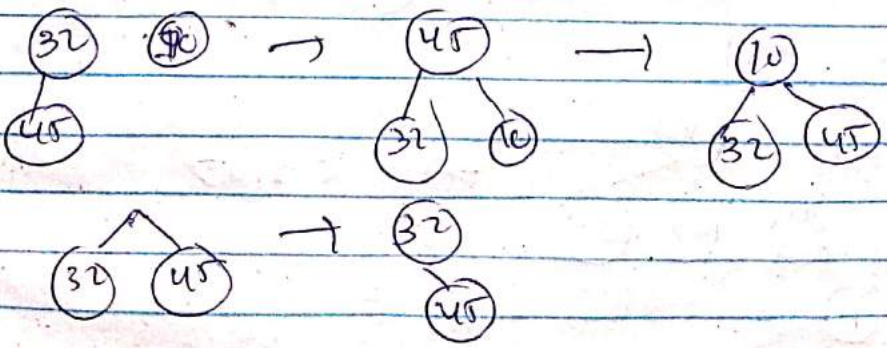
(4) 



(3) 



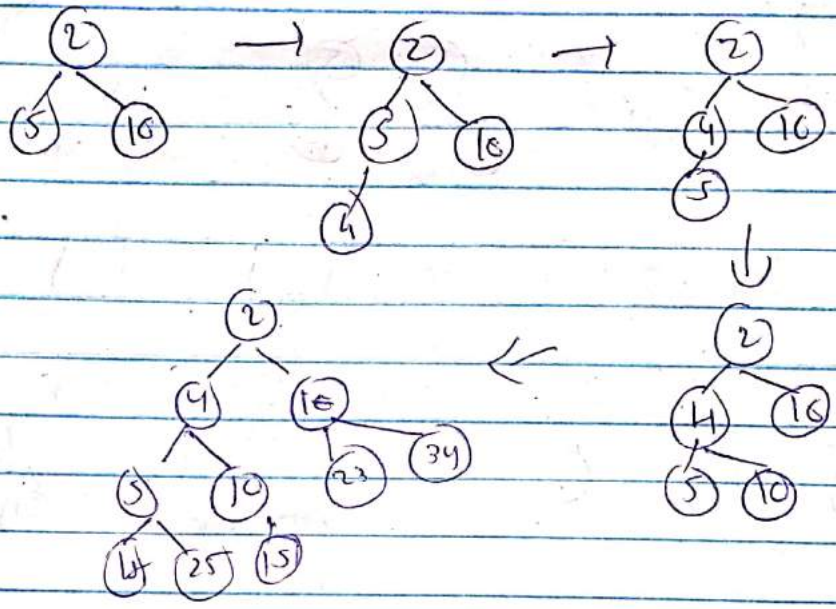
(6) 



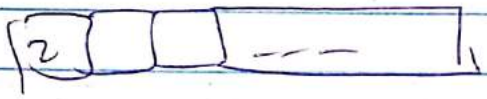
(9) 

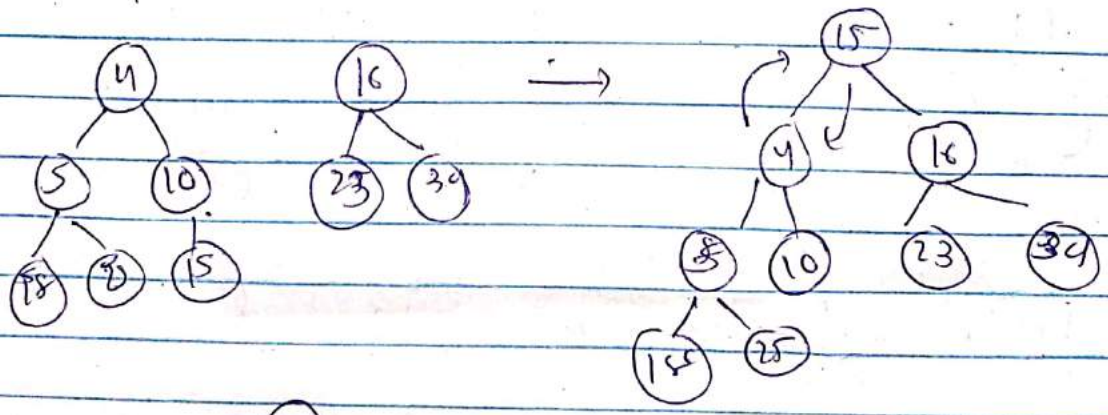
Sorted Array

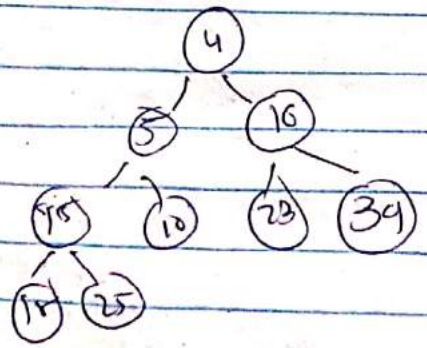
(2) < 3, 5, 16, 4, 10, 23, 34, 18, 25, 15 >



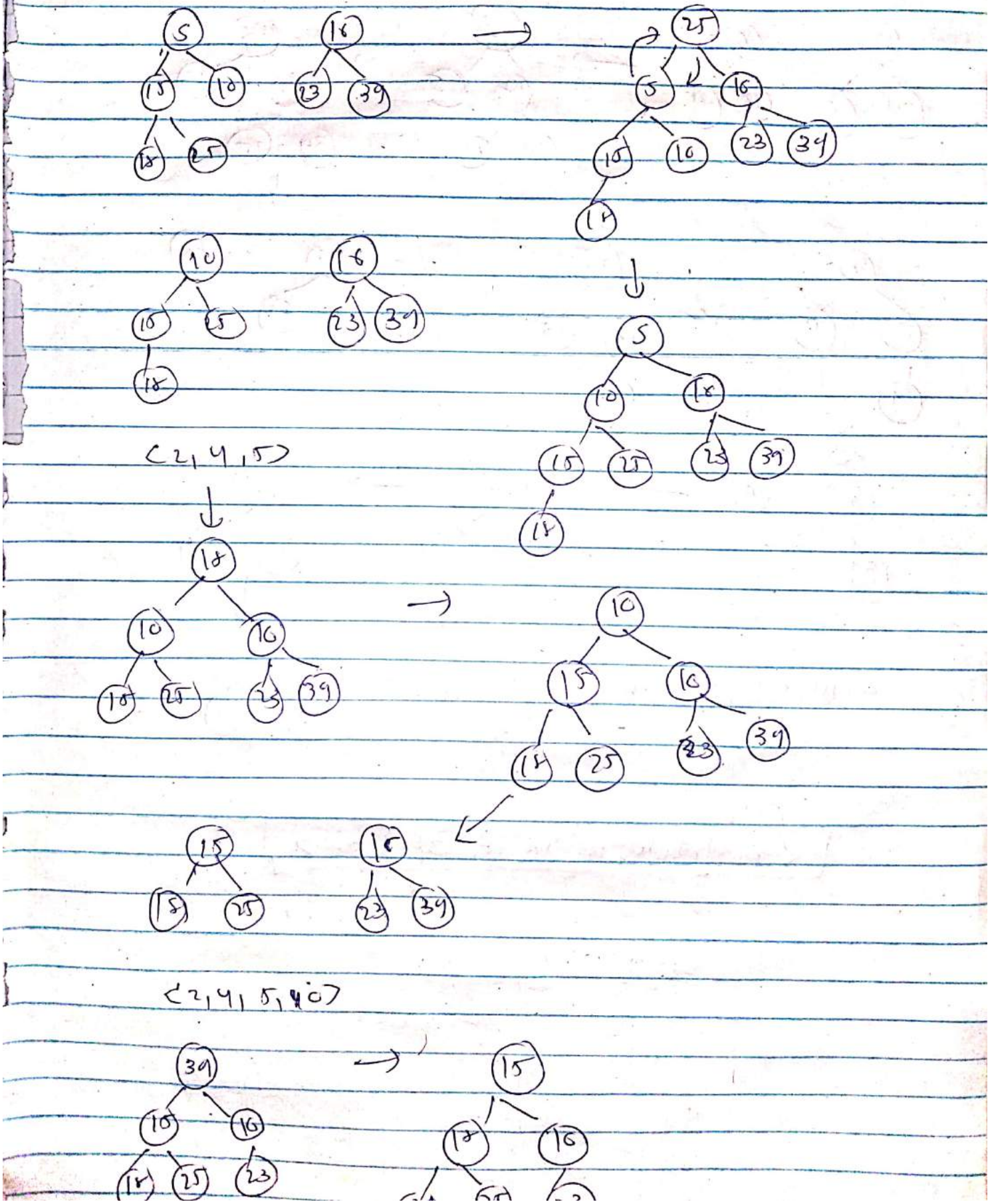
new heap

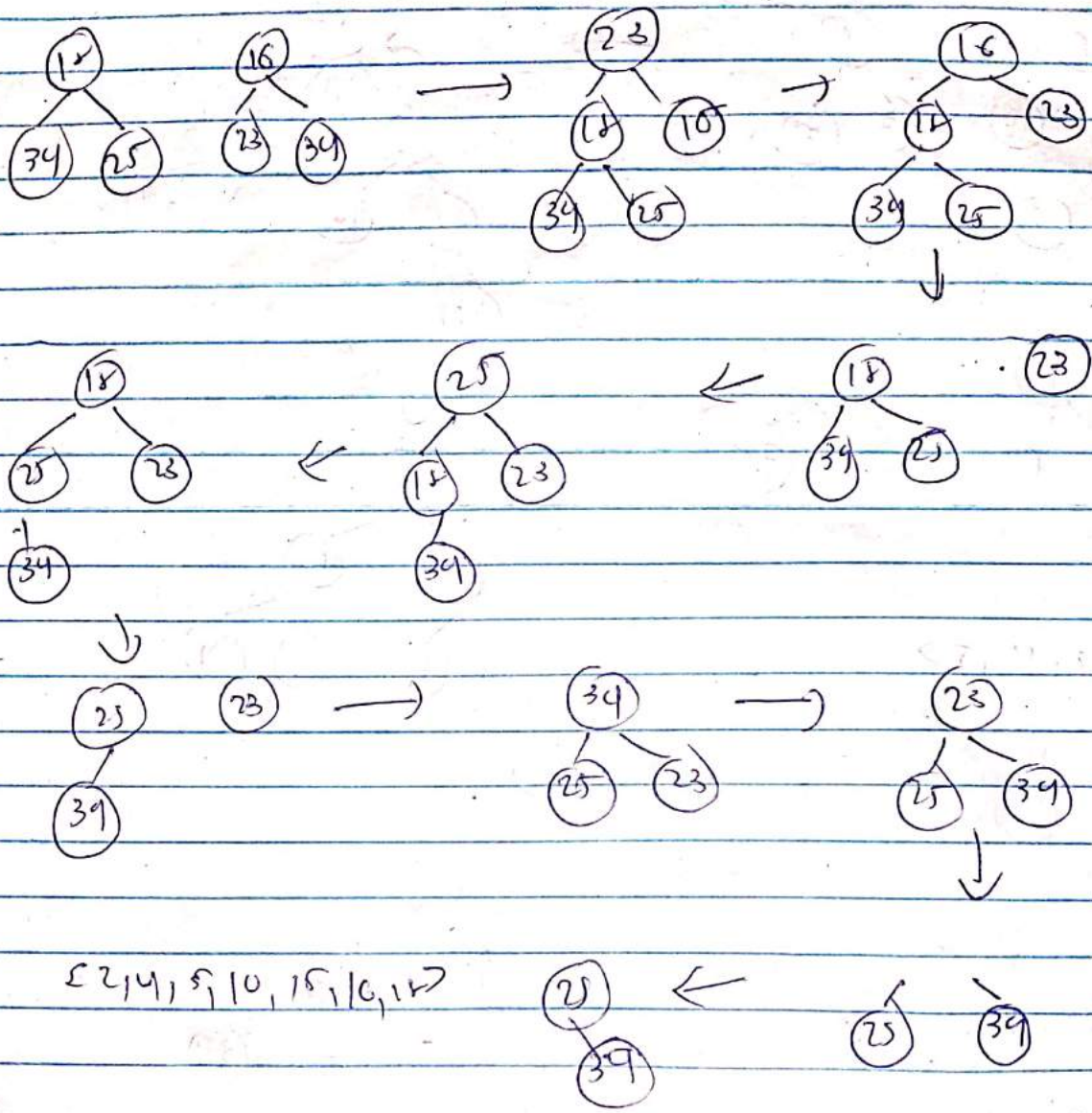






⇒ < 2, 4 >





{2, 4, 5, 10, 15, 16, 17}

Ans,

< 2, 4, 5, 10, 15, 16, 17, 25, 34 >

Name: 1901301070411

Tutorial:-3

① Define:-

ans ① feasible solution

A solution (set of value for the desicion variable) for which all the constraint in the solver model are satisfied it called a feasiable solution

② Optimal solution

→ An optional soution if festiable solution where the objective function reaches its maximum value

② Explain characteristic of greedy-method with suitable Example.

ans. greedy (D, n)

```
solution → 0
for ← 1 to n do
{
S ← select (D) // section of solution
if (feasible (solution, s)) then
    solution ← union (solution, s)
}
return solution
```

① first we select some solution from input domain

(3) from the set of feasible solution particular solution that satisfies or nearly satisfies the objectives of the function such a solution is called optimal solution

(3) Define:-

(1) spanning tree:-

- A spanning tree of a graph is a subgraph which is itself a tree and contains all the vertice of a connected no circuit.

(2) minimum spanning tree

⇒ A minimum spanning tree of a weighted connected graph is a spinning tree with minimum or smallest weight.

(3) connected graph:-

A graph in which we can visit from any one vertex to any other vertex. It called as a connected graph

(4) fully connected graph:-

→ It is a connected graph where unique edge connects each pair of vertices

(5) Dense graph:-

Dense graph is graph in which the number of edge is close to the maximal number of edges.

(6) sparse graph:

It is a graph in which the number of edges is close to the maximal number of edges sparse graph can be disconnected graph.

(4) Explain prim's algo & kruskal algo also write a comparison of both

Ans (1) prim's Algo.

→ Prim's algo. used to find the minimum spanning tree from a graph

step 1:- Select a starting vertex.

Step 2:- Repeat step 3 and 4 until then are formed vertex.

step3:- select and edge & connecting the tree vertex and fringe vertex that has minimum weight.

step 4:- Add the selected edge and the vertex to the minimum spanning tree T
(End loop)

step 5:- Exit.

(2) Kruskal's algorithm.

→ Kruskall's algorithm is used to find the minimum spanning tree for a connected weight graph

→ step 1:- Create a forest in such way that each graph is a separate tree

- step 2:- create a priority queue output connect all the edges of the graph

- step 3:- perform step-4 and which a is not empty

- step 4:- remove and edge from

the forest.
else
Discard the edge.

step 6:- End

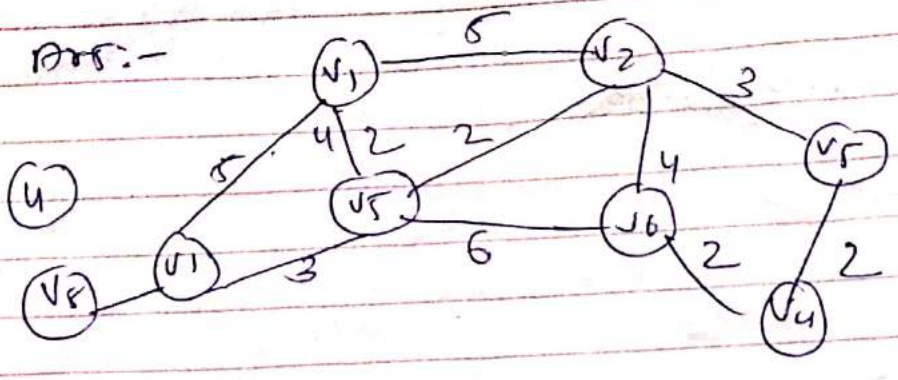
Prim's algorithm	Kruskal's Algorithm
→ It starts to build the minimum spanning tree from any vertex in the graph	It starts build the minimum spanning tree from the vertex carrying minimum cost in the graph
→ Prim's algorithm runs faster in dense graphs.	Kruskal's algorithm runs faster in sparse graph
→ Prim's algorithm used list dated structure.	Kruskal's algorithm used heap data structure.

(5) given an undirected graph as follows:-

$v_1 \cdot v_2 = 6$ $v_2 \cdot v_3 = 3$ $v_3 \cdot v_4 = 2$ $v_6 \cdot v_4 = 2$
$v_1 \cdot v_7 = 6$ $v_2 \cdot v_5 = 2$ $v_5 \cdot v_6 = 6$ $v_8 \cdot v_6 = 4$
$v_1 \cdot v_5 = 4$ $v_2 \cdot v_6 = 4$ $v_5 \cdot v_7 = 3$ $v_7 v_8 = 5$

Find MST using Prim's & Kruskal's algorithm.

Ans:-

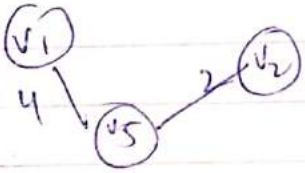


(1) Prim's algorithm:-

step 1:-

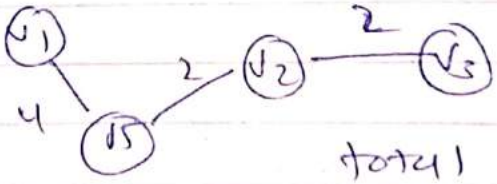
total weight = 4

step 2:-



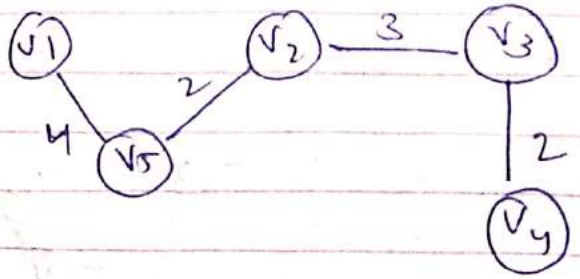
total weight = 6

step 3:-



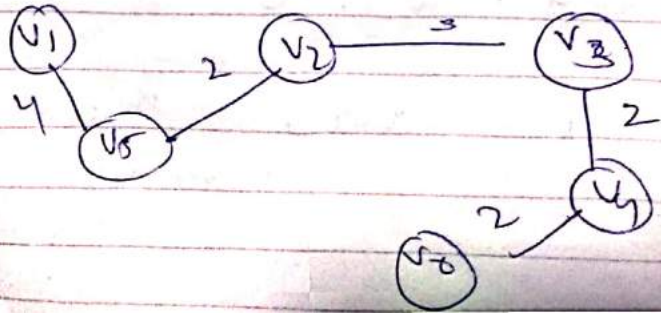
total weight = 9

step 4:-



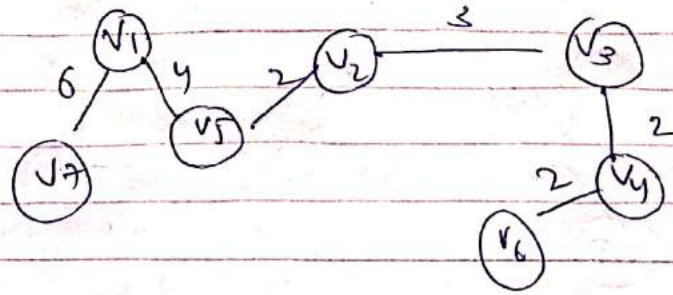
total weight=11

step 5:-



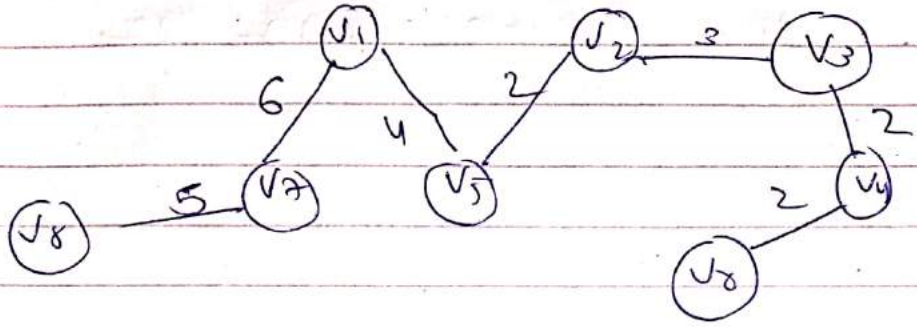
total weight = 15

Step 6:-



total weight = 24

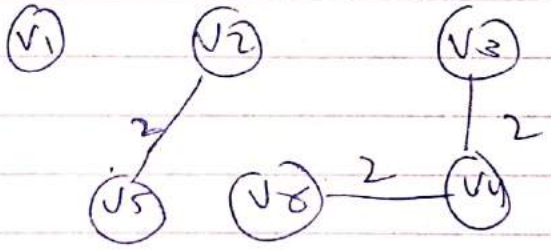
step 7:-



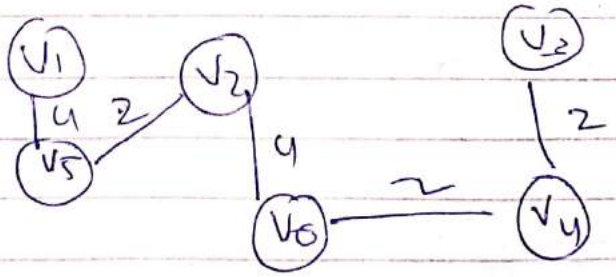
total weight = 24

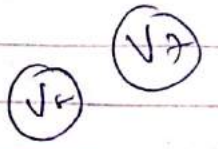
② Kruskal's algorithm:-

step 1:

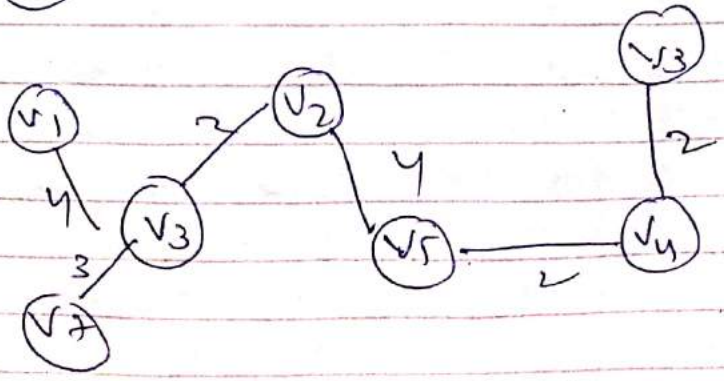


step 2:-

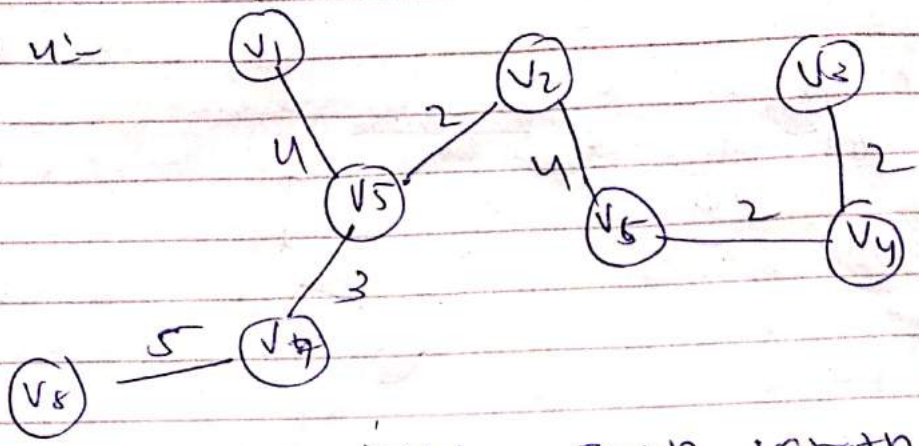




Step 3:-



Step 4:-



total path length = 22

~~190130107041~~

Tutorial :- 4

190130107041

* solve given binary knapsack problem using greedy method to get optimal profit.

1) $N=3$, $w=6$ $w(i) = (2,3,3)$ $v(i) = (1,2,4)$

2) $N=5$, $w=20$ $v(i) = (9,3,5,7,2)$.

$v(i) = (15,14,6,20,1$

Ans.

select object with maximum v/w ratio

v/w	x_1	x_2	x_3
	0.5	0.7	0.3

selected object	profit	weight	v/w	Remaining vlew
x_1	1	2	0.5	$6-2=4$
x_2	2	3	0.7	$4-3=$
x_3	$4 \times \frac{1}{2} = 2$	$3 \times \frac{1}{2} = 1.5$	1.3	$1-1=$

Hence solution is

$(x_1, x_2, x_3/2)$ with

total profit = 5

(2) $N=5$, $w=20$

$w(i) = (9, 3, 5, 7, 2)$

$v(i) = (15, 14, 6, 20, 10)$

object	1	2	3	4	5
v	15	14	6	20	10
w	9	3	5	7	2

selected object	profit	weight	remaining weight
x5	10	2	20-2 = 18
x4	20	7	18-2 = 11
x3	6	5	11-5 = 6
x2	14	3	6-3 = 3

total profit = 50
total weight = 17

(2) solve the given fractional knapsack problem to get optimal profit

N=5 w=35
w(i) = (10, 14, 12, 13, 15)
v(i) = (20, 35, 25, 40, 30)

Ans: Step 1: find the ration of value and weight.

item	weight	profit	p = v/w
1	10	20	2
2	14	35	2.5
3	12	25	2.1
4	7	40	5.7
5	15	30	2

step 2: arrange in order of decreasing

item	weight	profit	P = v/w
4	7	40	5.7
2	14	35	2.5
3	12	25	2.1
1	10	20	2
5	15	30	2

(4) compare greedy and dynamic programming

Diff.	greedy method	Dynamic method
	→ in greedy method sometime there is no such guarantee of getting optimal solution	It is guaranteed that Dynamic programming will generate an optimal solution as it generally considers all possible sequence the best.
	→ It is more efficient in term memory as it never look back or, revise previous choice	In recursive method for memoization it increase it's memory complexity.

→ A greedy method follows the problem solving heuristic of making the locally optimal choice at each stage

A dynamic programming is an algorithm technique which is usually based on a recurrent formula that used some previously calculated states.

(5) Explain the term principle of optimality.

Ans The principle of optimality is the basic principle of dynamic programming

→ which was developed by Richard Bellman that an optimal path has the property that whatever the initial condition and control variable over some initial period the control choose must be optimal for the remaining problem with the state resulting from the only decision taken to be the initial condition

Tutorial :- 5

1901301070411

1901301070411

Q:1 To calculated minimum number of coin where Demom invision are

$d_1 = 1$, $d_2 = 2$, $d_3 = 5$, $d_4 = 7$

i/j	0	1	2	3	4	5	6	7	8	9	10	11	12
$d_1 = 1$	0	1	2	3	4	5	6	7	8	9	10	11	12
$d_2 = 2$	0	1	1	2	2	3	3	4	4	5	5	6	6
$d_3 = 5$	0	1	1	2	2	1	2	2	3	3	4	4	5
$d_4 = 7$	0	1	1	2	2	1	2	1	3	2	3	3	2

→ There is 2 minimum coins required to make a change

→ Now for finding ways to make change use.

Ans.

i/j	0	1	2	3	4	5	6	7	8	9	10	11	12
$d_1 = 1$	1	1	1	1	1	1	1	1	1	1	1	1	1
$d_2 = 2$	1	1	2	2	2	3	4	4	5	5	6	6	7
$d_3 = 5$	1	1	2	2	3	4	5	6	7	8	10	11	13
$d_4 = 7$	1	1	2	2	3	4	5	7	8	10	12	14	17

→ There are 17 ways to set change of 12 by these valued coins.

Q.2 Hence we have given that

$$A^0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & \infty & \infty & \infty \\ \infty & 0 & \infty & 2 & 4 \\ \infty & 9 & 0 & 5 & 3 \\ \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & 0 \end{bmatrix} \end{array}$$

from these we will try A^1 to A^3

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & \infty & \infty & \infty \\ \infty & 0 & \infty & 2 & 4 \\ \infty & 9 & 0 & 5 & 3 \\ \infty & \infty & \infty & 0 & \infty \\ 2 & 3 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

$$A^2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & \infty & 3 & 5 \\ \infty & 0 & \infty & 2 & 4 \\ \infty & 9 & 0 & 5 & 3 \\ \infty & \infty & \infty & 0 & \infty \\ 2 & 3 & \infty & 5 & 0 \end{bmatrix} \end{array}$$

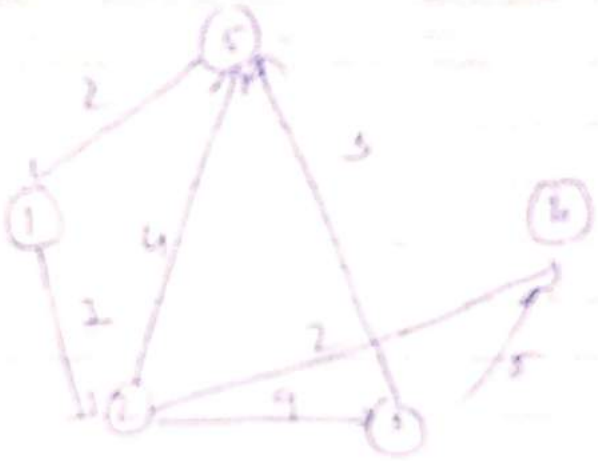
$$A^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & \infty & 3 & 5 \\ \infty & 0 & \infty & 2 & 4 \\ \infty & 9 & 0 & 5 & 3 \\ \infty & \infty & \infty & 6 & 2 \\ 1 & 3 & \infty & 3 & 0 \end{bmatrix} \end{array}$$

$$A^4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 1 & \infty & 3 & 5 \\ \infty & 0 & \infty & 2 & 4 \\ \infty & 9 & 0 & 5 & 3 \\ \infty & \infty & \infty & 0 & \infty \\ 2 & 3 & \infty & 5 & 0 \end{bmatrix} \end{array}$$

D^5 =

	1	2	3	4	5
1	0	-1	∞	3	5
2	6	0	∞	2	4
3	5	5	0	5	3
4	∞	∞	∞	0	∞
5	2	3	∞	5	0

[illegible] DT chart [illegible] with [illegible] graph is

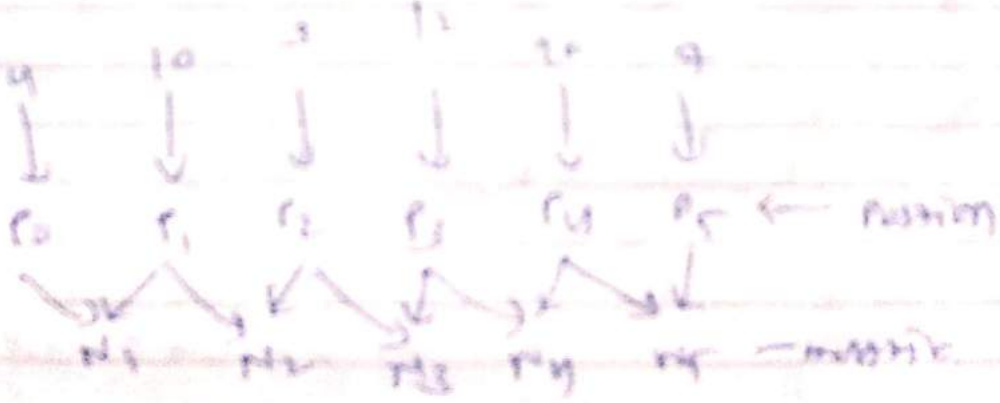


Q:3 [illegible]

P = 4x10 D = 12x10

S = 10x3 L = 3x7

C = 3x12



→ product of 2 matrix:-

(1) $m(1,2) = m_1 \times m_2$

$= 4 \times 10 \times 10 \times 3$

$= 120$

(2) $m(2,3) = m_2 \times m_3$

$= 10 \times 3 \times 3 \times 12$

$= 10 \times 3 \times 12 = 360$

3) $m(3,4) = m_3 \times m_4$

$= 3 \times 12 \times 12 \times 20$

$= 3 \times 12 \times 20$

$= 720$

(4) $m(4,5) = m_4 \times m_5$

$= 12 \times 20 \times 7$

$= 1680$

→ product of 3 matrices:

(1) $m(1,3)$:

$(m_1 \times m_2 \times m_3)$

$= \min \begin{cases} m(1,2) + m(3,3) + p_0 r_2 p_3 \\ = 120 + 0 + (4 \times 3 \times 12) \\ = 264 \\ m(1,1) + m(2,3) + p_0 r_1 p_3 \\ 0 + 360 + (4 \times 10 \times 12) \\ = 840 \end{cases}$

$m(1,3) = 264$

② $m(2,4)$

$= (M_2 \times M_3 \times M_4)$

$= \min \begin{cases} m(2,3) + m[4,4] + P_1 P_3 P_4 \\ = 360 + 0 + (10 \times 12 \times 20) \\ \qquad 2760 \\ M[2,2] + m[3,4] + r_1 P_2 P_4 \\ = 0 + 720 + (10 \times 3 \times 20) \\ \qquad = 1320 \end{cases}$

$M(2,4) = 1320$

⇒ Let's take product of 4 matrices

$M[1,4] = (m_1 \times m_2 \times m_3 \times m_4)$

I) $M[1,4] =$

$\min \begin{cases} m[1,3] + m[4,4] + P_0 P_3 P_4 \\ = 264 + 0 + (4 \times 12 \times 20) \\ \qquad = 1224 \\ M[1,2] + m[3,4] + P_0 P_2 P_4 \\ = 120 + 720 + (4 \times 3 \times 20) = 1080 \\ M[1,1] + M[2,4] + P_0 P_1 P_4 \\ = 0 + 1320 + 4 \times 10 \times 20 = 2120 \end{cases}$

$M[1,4] = 1080$

2) $M[2,5] = (m_2 \times m_3 \times m_4 \times m_5)$

$m(2,5) = \min \begin{cases} m(2,4) + m(3,5) + P_1 P_4 P_5 \\ = 1320 + 0 + (10 \times 20 \times 7) = 2720 \\ M[2,3] + m[4,5] + P_1 P_3 P_5 \\ = 360 + 1080 + (10 \times 12 \times 7) \\ \qquad = 2280 \end{cases}$

m[1,2] + m[3,5] + $p_1 p_2 p_3$
0 + 1140 + (10 × 3 × 7)
= 1350

m(2,5) = 1350

→ Least take product of 5 matrices

→ m[1,5] = ($m_1 \times m_2 \times m_3 \times m_5$)

m(1,5) = min {
m[1,4] + m[5,5] + $p_0 p_4 p_5$
= 1080 + 0 + (4 × 20 × 7 = 1544

m[1,3] + m[4,5] + $p_0 p_3 p_5$
= 205 + 1680 + (4 × 12 × 7)
= 2016

m[1,2] + m[3,5] + $p_0 p_2 p_5$
= 120 + 1140 + (4 × 3 × 7)
= 1344

m[1,1] + m[2,5] + $p_0 p_1 p_5$
= 0 + 1350 + (4 × 10 × 7)
= 1630
}

m[1,5] = 1344

→ Output is

1	2	3	4	5	
0	120	264	1050	1344	1
	0	380	1320	1350	2
		0	720	1140	3
			0	1650	4
				0	5

- so order will be first (M3 x M4 x M5) then (M1 x M2) x (M3 x M4 x M5)

Q:5 S1 = A, B, C, D, B, A, C, D, F

S2 = C, B, A, F

S1	S2	A	C	B	A	F
A	0	0	0	0	0	0
A	0	← 0	← 0	← 1	← 1	← 1
B	0	← 0	← 1	← 1	← 1	← 1
C	0	← 1	← 1	← 1	← 1	← 1
D	0	↑ 1	← 1	← 1	← 1	← 1
B	0	↑ 1	← 2	← 2	← 2	← 2
A	0	↑ 1	← 2	← 3	← 3	← 3
C	0	↑ 1	↑ 2	↑ 3	↑ 3	
D	0	↑ 1	↑ 2	↑ 3	↑ 3	
F	0	↑ 1	↑ 2	↑ 3	↖ 4	
		C	B	A	F	

→ from last box of table we get know that longest common subsequence have length of 4

→ from table we get that longest common subsequence is

CBAF

Nikhil

190130707041

Tutorial - 6

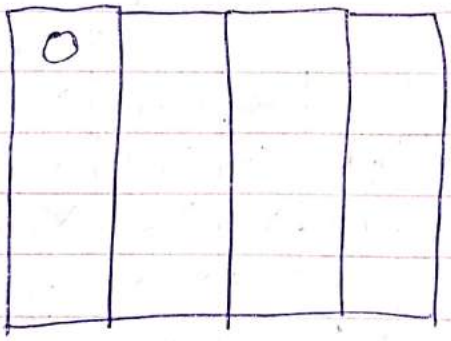
190130107041

1) Define backtracking

→ Backtracking can be defined as a general algorithmic technique that considers searching every possible combination in order to solve a computational problem.

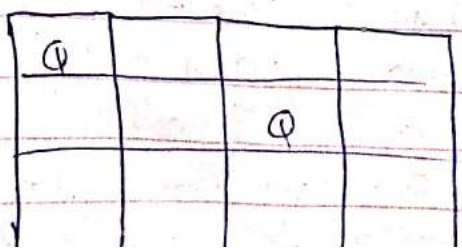
2) Solve 4 queen problem using backtracking

→ - Now we start with empty chessboard
- Place queen 1 in the first possible position of it's row ie 1st row 1st column.

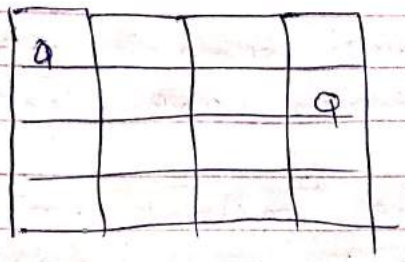


- then place queen 2 after trying unsuccessful place (1,2), (2,1), (2,2), at (2,3)

i.e. 2nd row and 3rd column

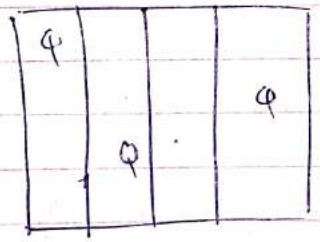


1)



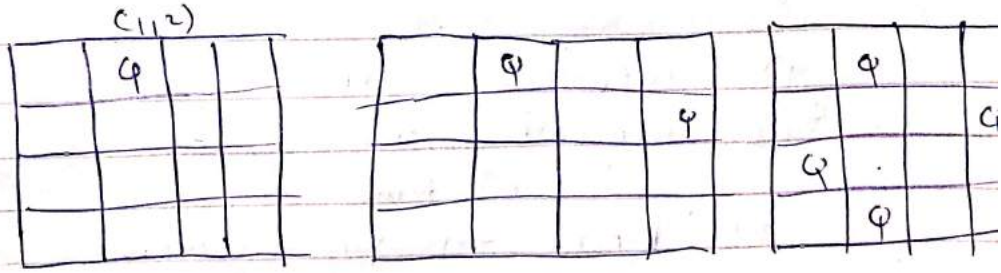
→ This is the dead end because a 3rd queen can not be placed in next column. as there is no acceptable position for queen 3 Hence. algo backtracking and the placed the 2nd queen at (2,4) position.

→



The place 3rd queen at (3,2) but is again another dead end as next queen can not be placed at permissible position

→ Hence we need backtrack till the column up to queen and move it to.



This situation is obtained (3,4,1,3) in row wise manner

(3) Explain rabin karp method for string matching and also give the algorithm

→ The rabin karp method is based on hashing technique. This also assumes each character to be a digit. in radixed notation. It makes use of equivalence of two number modulo a third number. Let P[1--m] be a pattern then denote its corresponding decimal value Then d=10 for decimal number d=2 for binary number.

Similarly

Let T[1--n] be a text then is denoted the decimal value of the substring T[s+1] for s=0; 1; n-m

The method follows following steps

1) compute p in O(m) time
2) compute all value in total of O(n) time.
3) Find all valid shift in total of O/P time.

for example:- To compute pattern

P[1-m] = 41603 we have m=5 d=10

then

$p = P[m] + 10[P[m-1]] + 10^2(P[m-2]) + \cdots + 10^{n-1}[n]$

we can then compute to form $T[s-m]$ in $O(m)$ time then remining ti can be computed in $O(n-m)$ thime or

$T_{s+1} = d[t_s - d^{m-1} T[s+0] + T[s+m+1]]$

where $d=10$

Algorithm rabin kamp [T(1..n], P[1..m], q]

// problem descripit this algo is for matching the pattern wirg Rabin - karp method

```
h ← pow (d, m-1) mod q ←
P ← 0
to ← 0
for ( i ← 1 to m)
{
P ← (dp + P[i]) mod q
to ← (dto + T[i]) mod q
}                                   } O(m) time
for (s ← 0 to n-m)
{ if (P = to) then
{
if (q(1.m)) = T[s+1 .. s+m]
then
write (pattern fount with shift)
}
if (s<(n-m))
t_{s+1} ← (t_s - T[T+1] ht .
T[s+m+1] ode
}
```

(4) Explain finite automate for string matching

→ finite automate is a very effective tool it used in string matching algorithm.

→ matching the pattern in this manner makes the process of string matching very efficient

→ In this method each character of the the text is mated exactly once

→ Hence O(n) time is required for Examin the text character

→ But in this algorithm a large amount of time it needed to build finite automatic

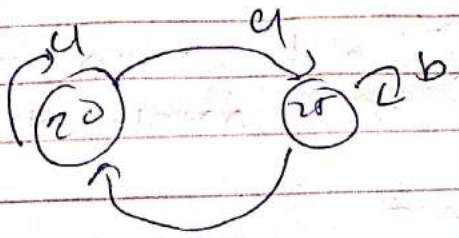
→ Befor learing the actual string matching algorithm let us go throughn some basic concepts of finite automate

→ Definition of finit Automate:-

→ A finte automation it collection of

(1) Q a finatie set of states
(2) $q_0 \leftarrow Q$ is a start state
(3) $qE \leftarrow Q$ is a Accept state or
(4) Σ is a finit input set
5. δ a mapping function $Q * \Sigma$

for example



State	a	b
q0	q1	q0
q1	q0	q2

transition table

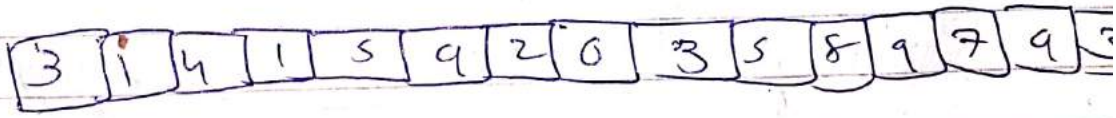
→ Here q0 is start state and q1 is a final state the above design finite automate it for accepting a language which consists of odd number of a's and any number of b's

(5) for string matching working

q = 11

pattern [2 6]

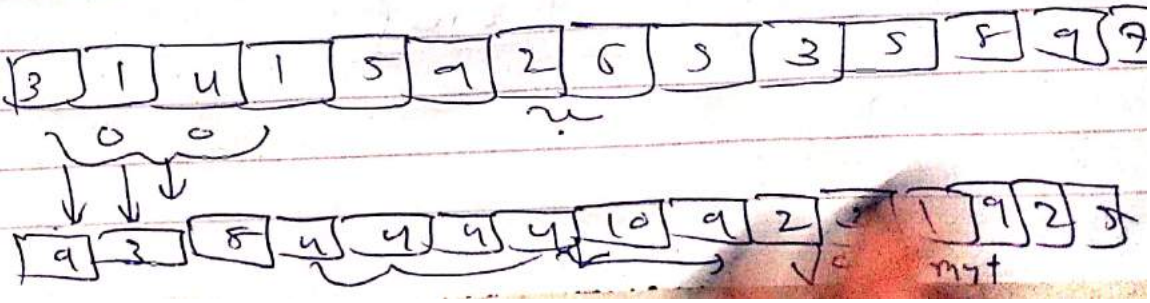
text:



the 20 mod 11 = 4

we will now obtain mod 11 value for each text.

Step:-



for m=2 consider $t_s = 26$ then we can compute t_{s+1} as follows

t_{s+1} = 10 (t_s − d^{m−1} old high order digit × new low order digit)

$= 10\,(26 - 10 \times 2) + 5$

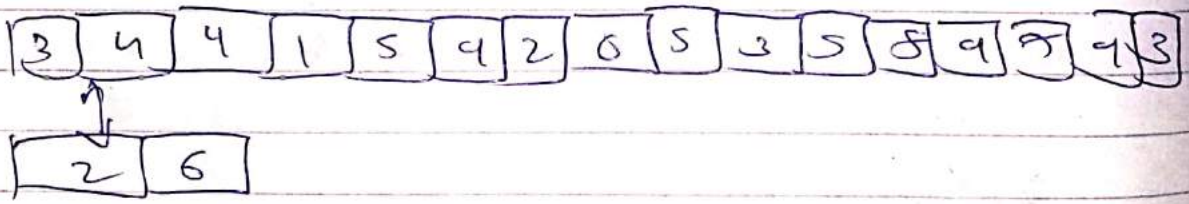
$= 10(26-20) + 5$

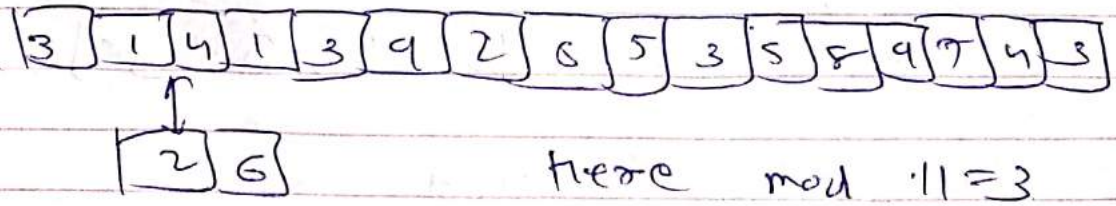
$= 60 + 5$

$t_{s+1} = 65$

→ thus we can obtain every successive bit text.

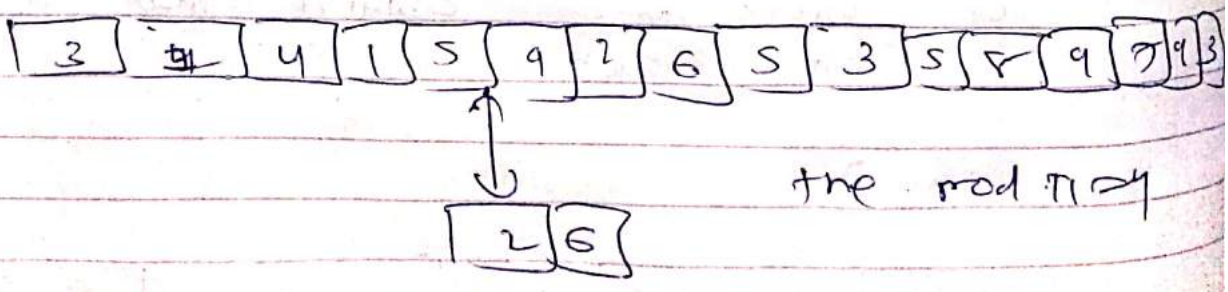
→ According to Rabin-Karp method we start matching the pattern text from beginning it self



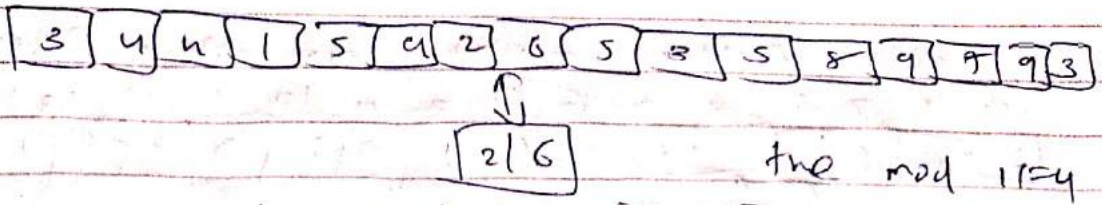


Here mod 11 = 3

hence move to right



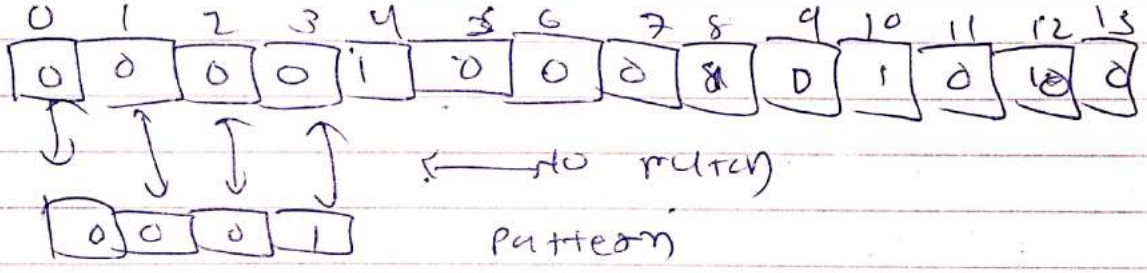
the mod 11 = 4



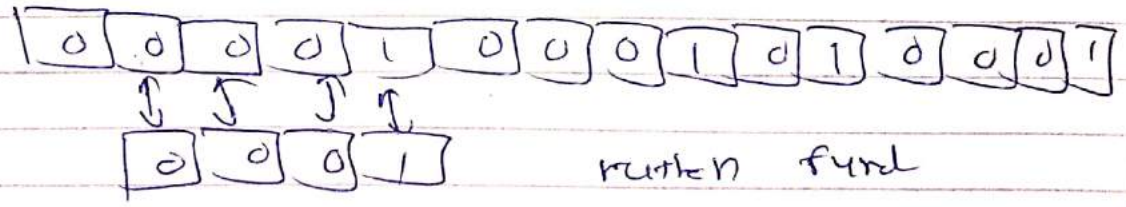
the mod 11=4 and putter is also matching against it hence we declare that at 4th position we find pattern matching

(6) show the comparisons the exetive strin matches makes for the Putter P= 0001 in the text T=0000. 1000.1010001

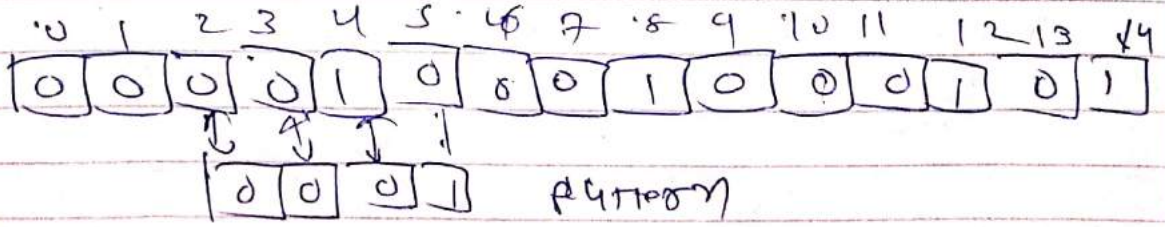
→ Let



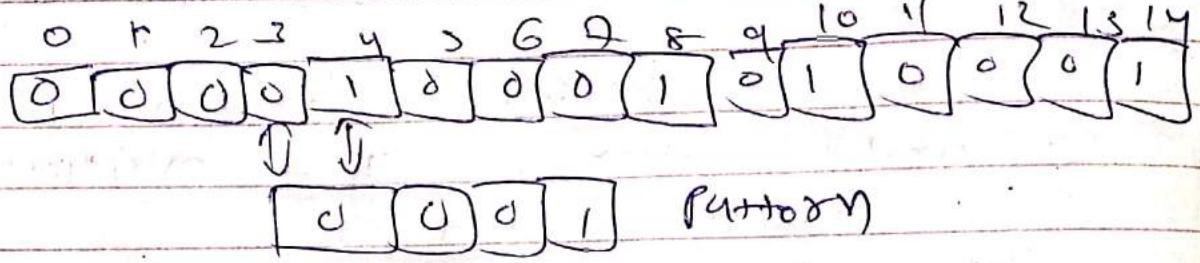
→ shift to right by one position



→



→ shift pattern right by one position



→ similarly